Industrial Major Project Report

on

Client Server Model Implementation In Docker Containers

submitted in fulfilment of the

requirements for the award of the degree of Bachelor of Technology

in

Department of Computer Science and Engineering.

by

Venkatapathi Amulya	**13241A05N8**
Sreeram Mounika	**13241A05M2**
Vangala Ushasree	**13241A05N4**
Aishwarya Varala	**13241A0505**

Under the Guidance of

Dr.G.R.Sakthidharan

Professor

Department of Computer Science and Engineering

GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND TECHNOLOGY

(Autonomous under JNTUH, Hyderabad)

Bachupally, Kukatpally,Hyderabad- 500090

2016-17

Department of Computer Science and Engineering

GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND TECHNOLOGY

(Autonomous under JNTUH, Hyderabad)

Bachupally, Kukatpally Hyderabad- 500090

CERTIFICATE

This is to certify that the Industrial major project entitled "**Client Server Model Implementation In Docker Containers**" is submitted by **Venkatapathi Amulya, Sreeram Mounika, Vangala Ushasree, Aishwarya Varala (13241A05N8, 13241A05M2, 13241A05N4,13241A0505)** in fulfilment of the requirement for the award of the degree in Bachelor of Technology in Computer Science and Engineering during academic year 2016-2017.

Internal Guide	**Head of the Department**
Dr.G.R.Sakthidharan	Dr. Ch. Mallikarjuna Rao
Professor	Professor
TCS TBU Head	**Mentor**
Kotaru Kiran	Naveen

External Examiner

DECLARATION

We here by declare that the industrial major project entitled **“Client Server Model Implementation In Docker Containers”** is submitted in the fulfillment of the requirements for the award of degree of Bachelor of technology in Computer Science and Engineering from Gokaraju Rangaraju Institute of Engineering and Technology (Autonomous under Jawaharlal Nehru Technology University, Hyderabad). The results embodied in this project have not been submitted to any other university or Institution for the award of any degree or diploma.

Venkatapathi Amulya
(13241A05N8)
Sreeram Mounika
(13241A05M2)
Vangala Ushasree
(13241A05N4)
Aishwarya Varala
(13241A0505)

ACKNOWLEDGEMENT

There are many people who helped us directly and indirectly to complete our project successfully. We would like to take this opportunity to thank one and all.

First of all we would like to express our deep gratitude towards our internal guide Dr.G.R.Sakthidharan,Professor,Department of Computer Science and Engineering for his support in the completion of our dissertation. We wish to express our thanks to Dr.Ch.Mallikarjuna Rao,HOD,Department of Computer Science and Engineering and also to our Principal Dr.Jandhyala N.Murthy for providing the facilities to complete the dissertation.We would like to thank the college management for the support in the completion of our dissertation.

We would like to thank all our faculty and friends for their help and constructive criticism during the project period. Finally we are very much indebted to our parents for their moral and financial support and encouragement to achieve goals.

Venkatapathi Amulya
(13241A05N8)
Sreeram Mounika
(13241A05M2)
Vangala Ushasree
(13241A05N4)
Aishwarya Varala
(13241A0505)

ABSTRACT

Generally for an application to run on an operating system, it first needs to know the dependencies of the platform on which it is deployed on. Virtualization concept solves this problem. Virtualization means to create a virtual version of a device or a resource such as operating system where the framework divides the resource into one or more execution environments.

Installing a hypervisor on the host operating system increases the overhead and sharing of resources will be difficult. This problem can be overcome by docker containers.Docker allows you to package an application with all of its dependencies into a standardized unit for software development. Docker containers wrap up a piece of software in a complete file system that contains everything it needs to run: code, runtime, system tools, libraries –anything you can install on a server. This guarantees that it will always run the same, regardless of the environment it is running in.

In our project, Client server application is developed using python scripting language. The client and server communicate with each other by sending sample messages. The data transferred is printed along with time. To make this application to run on any platform without any overload, the application is to be deployed in docker.

LIST OF FIGURES

LIST OF TABLES

CONTENTS **PAGE NOS**

CHAPTER-3

DOCKER

CHAPTER-4

DEPLOYING PYTHON APPLICATION IN DOCKER CONTAINERS

CHAPTER-1

CLIENT SERVER MODEL IMPLEMENTATION IN DOCKER CONTAINERS

1.1 INTRODUCTION

` Client and server application should be developed in python and establish communication between them with sample messages along with time. Server and client application should run inside a docker container. Seamlessly they should be able to communicate with each other. Whenever there is communication the data transferred should be printed.

1.2 NUMBER OF MODULES

There are two modules in client and server model implementation using python and deploy in docker containers. They are:

1. Client server implementation in python
2. Deploy and run client server application inside the docker containers.

- **Module 1:**

  Client server implementation in python:

  Client and Server application should be developed in python and establish a communication between them. In addition to it, Server and client should generate logs when a communication happens between them (Like logging of time).

- **Module 2:**

  Deploy and run client server application inside the docker containers:

  Containerize the server and client applications into two containers using Docker. Launch the two containers and the containers should be up and running with proper communication.

1.3 SYSTEM REQUIREMENTS

Minimum software requirements:

- ➢ Docker tool box
- ➢ Python 3.6.0
- ➢ Windows 8/10 operating system

Minimum hardware requirements:

- ➢ 4GB RAM
- ➢ 250GB harddisk
- ➢ i3 processor

CHAPTER-2

PYTHON

2.1 WHAT IS PYTHON

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL).

Python is amazingly portable and can be used in almost all operating systems. Python is interpreted and is easy to extend. You can extend Python by adding new modules that include functions, variables, or types through compiled C or C++ functions. You can also easily embed Python within C or C++ programs, allowing you to extend an application with scripting capabilities. One of the most useful aspects of Python is its massive number of extension modules. These modules provide standard functions such as string or list processing, but there are also application-layer modules for video and image processing, audio processing, and yes, networking. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

Fig 2.1.1 Python logo

- **Python is Interpreted:**

  Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.

- **Python is Interactive:**

  You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

- **Python is Object-Oriented:**

  Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

- **Python is a Beginner's Language:**

  Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

2.2 HISTORY OF PYTHON

- Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.
- Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages.
- Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).
- Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

2.3 PYTHON FEATURES

- **Easy-to-learn:** Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read:** Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain:** Python's source code is fairly easy-to-maintain.
- **A broad standard library:** Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode:** Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.

- **Portable:** Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable:** You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases:** Python provides interfaces to all major commercial databases.
- **GUI Programming:** Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable:** Python provides a better structure and support for large programs than shell scripting.

Python has a big list of good features, few are listed below:

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- IT supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

2.4 WHY WE USE PYTHON?

The number one reason to learn and use Python is its popularity. The size of its user base and the growing number of applications built with Python make it a worthwhile investment.You find Python in several development areas -- it's used to build system utilities, as a glue language for program integration, for Internet applications, and for rapid prototyping.

Python also has some advantages over other scripting languages. It has a simple syntax and is conceptually clear, making it easy to learn. Python is also easier and more descriptive when using complex data structures (such as lists, dictionaries, and tuples). Python can also extend languages and be extended by languages in turn.

2.5 LOCAL ENVIRONMENT SETUP

Open a terminal window and type "python" to find out if it is already installed and which version is installed.

- Unix (Solaris, Linux, FreeBSD, AIX, HP/UX, SunOS, IRIX, etc.)
- Win 9x/NT/2000
- Macintosh (Intel, PPC, 68K)
- OS/2
- DOS (multiple versions)
- PalmOS
- Nokia mobile phones
- Windows CE
- Acorn/RISC OS
- BeOS
- Amiga
- VMS/OpenVMS
- QNX
- VxWorks
- Psion
- Python has also been ported to the Java and .NET virtual machines

2. 5. 1 Getting Python

The most up-to-date and current source code, binaries, documentation, news, etc., is available on the official website of Python https://www.python.org/. You can download Python documentation from https://www.python.org/doc/. The documentation is available in HTML, PDF, and PostScript formats.

2. 5. 2 Installing Python

Python distribution is available for a wide variety of platforms. You need to download only the binary code applicable for your platform and install Python.

If the binary code for your platform is not available, you need a C compiler to compile the source code manually. Compiling the source code offers more flexibility in terms of choice of features that you require in your installation.

Here is a quick overview of installing Python on various platforms −

- **Unix and Linux Installation**

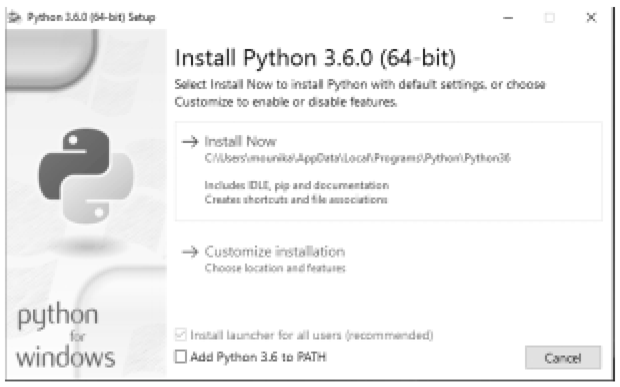
  Here are the simple steps to install Python on Unix/Linux machine.

  1. Open a Web browser and go to https://www.python.org/downloads/.
  2. Follow the link to download zipped source code available for Unix/Linux.
  3. Download and extract files.
  4. Editing the Modules/Setup file if you want to customize some options.
  5. run ./configure script
  6. make install
  7. This installs python at standard location /usr /local/bin.

- **Windows Installation**

  INSTALLATION STEPS:

  Step1: Click on Install Now.



Python 3.6.0 (64-bit) Setup

Install Python 3.6.0 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

→ Install Now
C:\Users\mounika\AppData\Local\Programs\Python\Python36
Includes IDLE, pip and documentation
Creates shortcuts and file associations

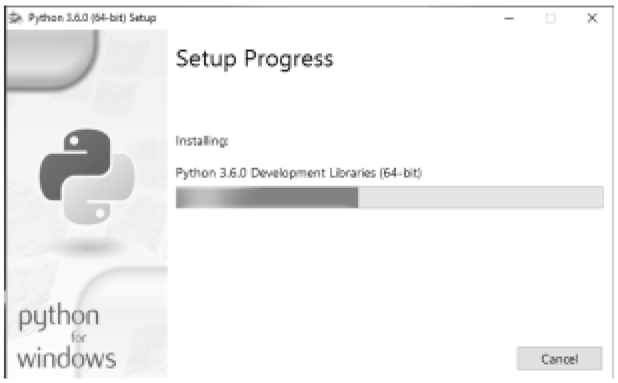
→ Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☐ Add Python 3.6 to PATH

Cancel

Fig 2.5.2.1 Python installation step1.

Step 2: Installing



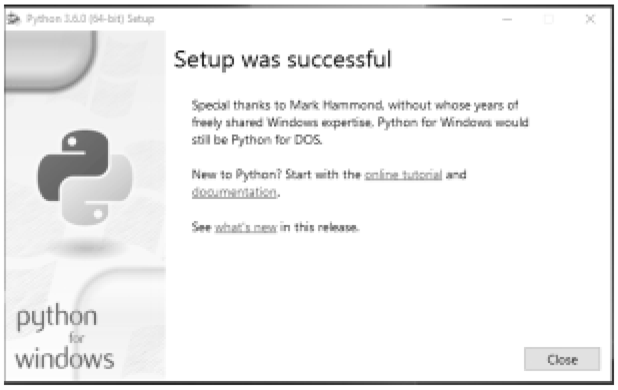
Setup Progress

Installing:

Python 3.6.0 Development Libraries (64-bit)

Fig 2.5.2.2 Python installation step 2

Step3: Successfully finished .Click on close.



Setup was successful

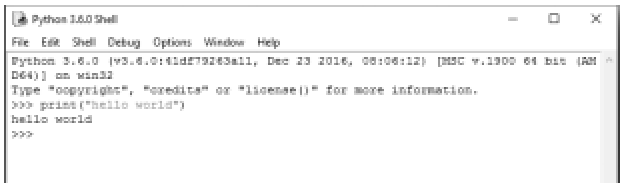
Special thanks to Mark Hammond, without whose years of freely shared Windows expertise, Python for Windows would still be Python for DOS.

New to Python? Start with the online tutorial and documentation.

See what's new in this release.

Fig 2.5.2.3 Python installation step 3

Python IDLE: This is used for executing Python programs.



```
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 08:06:12) [MSC v.1900 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print("hello world")
hello world
>>>
```

Fig 2.5.2.4 Python IDLE

➢ Macintosh Installation

Recent Macs come with Python installed, but it may be several years out of date. See http://www.python.org/download/mac/ for instructions on getting the current version along with extra tools to support development on the Mac. For older Mac OS's before Mac OS X 10.3 (released in 2003), MacPython is available.

Note: Jack Jansen maintains it and you can have full access to the entire documentation at his website − http://www.cwi.nl/~jack/macpython.html. You can find complete installation details for Mac OS installation.

2.5.3 Setting up PATH

Programs and other executable files can be in many directories, so operating systems provide a search path that lists the directories that the OS searches for executables.The path is stored in an environment variable, which is a named string maintained by the operating system. This variable contains information available to the command shell and other programs.The path variable is named as PATH in Unix or Path in Windows (Unix is casesensitive; Windows is not).In Mac OS, the installer handles the path details. To invoke the Python interpreter from any particular directory, you must add the Python directory to your path.

2.5.3.1 Setting path at Unix/Linux

To add the Python directory to the path for a particular session in Unix −

- **In the csh shell** − type sentence PATH "$PATH:/usr/local/bin/python" and press Enter.
- **In the bash shell (Linux)** − type export PATH="$PATH:/usr/local/bin/python" and press Enter.
- **In the sh or ksh shell** − type PATH="$PATH:/usr/local/bin/python" and press Enter.

2.5.3.2 Setting path at Windows

To add the Python directory to the path for a particular session in Windows −

- **At the command prompt** − type path %path%;C:\Python and press Enter.

Note − C:\Python is the path of the Python directory.

2. 6 PYTHON ENVIRONMENT VARIABLES

Here are important environment variables, which can be recognized by Python −

S.No.	Variable & Description
1	**PYTHONPATH** It has a role similar to PATH. This variable tells the Python interpreter where to locate the module files imported into a program. It should include the Python source library directory and the directories containing Python source code. PYTHONPATH is sometimes preset by the Python installer.
2	**PYTHONSTARTUP** It contains the path of an initialization file containing Python source code. It is executed every time you start the interpreter. It is named as .pythonrc.py in Unix and it contains commands that load utilities or modify PYTHONPATH.
3	**PYTHONCASEOK** It is used in Windows to instruct Python to find the first case-insensitive match in an import statement. Set this variable to any value to activate it.
4	**PYTHONHOME** It is an alternative module search path. It is usually embedded in the PYTHONSTARTUP or PYTHONPATH directories to make switching module libraries easy.

Table 2.6 Python environment variables

2.7 RUNNING PYTHON

There are three different ways to start Python −

- **Interactive Interpreter**

  You can start Python from Unix, DOS, or any other system that provides you a command-line interpreter or shell window. Enter **python** the command line. Start coding right away in the interactive interpreter.

  $python # Unix/Linux

  or

  python% # Unix/Linux (or) C:> python # Windows/DOS

  Here is the list of all the available command line options −

S.No.	Option & Description
1	**-d -** It provides debug output.
2	**-O -** It generates optimized byte code (resulting in .pyo files).
3	**-S -** Do not run import site to look for Python paths on startup.
4	**-v -** verbose output (detailed trace on import statements).
5	**-X -** disable class-based built-in exceptions (just use strings); obsolete starting with version 1.6.
6	**-c cmd -** run Python script sent in as cmd string
7	**File -** run Python script from given file

Table 2.7 Command line options

- **Script from the Command-line**

A Python script can be executed at command line by invoking the interpreter on your application, as in the following −

```
$python script.py # Unix/Linux

or

python% script.py # Unix/Linux

or

C: >python script.py # Windows/DOS
```

Note − Be sure the file permission mode allows execution.

- **Integrated Development Environment**

You can run Python from a Graphical User Interface (GUI) environment as well, if you have a GUI application on your system that supports Python.

1. **Unix** − IDLE is the very first Unix IDE for Python.
2. **Windows** – Python Win is the first Windows interface for Python and is an IDE with a GUI.
3. **Macintosh** − The Macintosh version of Python along with the IDLE IDE is available from the main website, downloadable as either MacBinary or BinHex'd files.

If you are not able to set up the environment properly, then you can take help from your system admin. Make sure the Python environment is properly set up and working perfectly fine.

2.8 SOCKET PROGRAMMING IN PYTHON

Python offers two basic sockets modules. The first, Socket, provides the standard BSD Sockets API. The second, Socket Server, provides a server centric class that simplifies the development of network servers. It does this in an asynchronous way in which you can provide plug-in classes to do the application-specific jobs of the server.

Python provides two levels of access to network services. At a low level, you can access the basic socket support in the underlying operating system, which allows you to implement clients and servers for both connection-oriented and connectionless protocols.

Python also has libraries that provide higher-level access to specific application-level network protocols, such as FTP, HTTP, and so on.

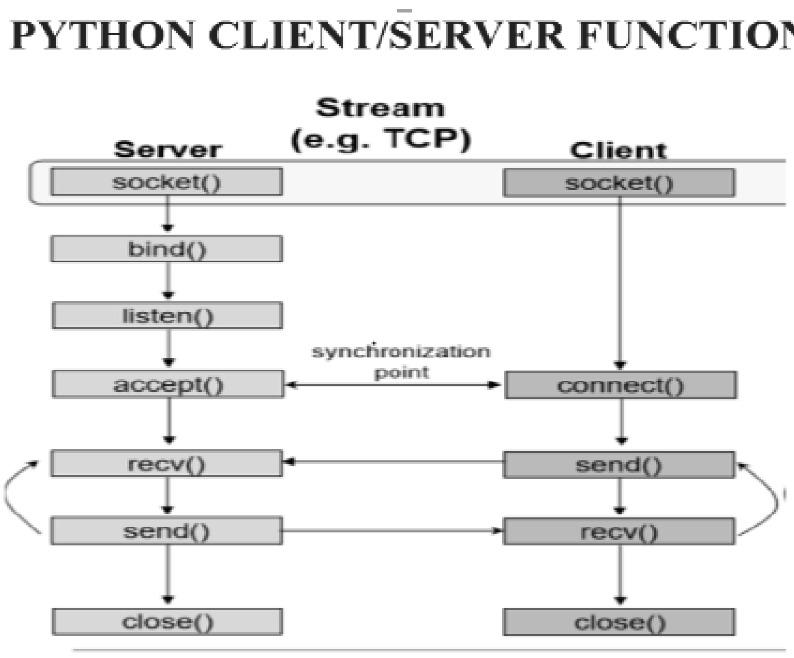


Fig 2.8 Python client/server functions

What are Sockets?

Sockets are the endpoints of a bidirectional communications channel. Sockets may communicate within a process, between processes on the same machine, or between processes on different continents.

Sockets may be implemented over a number of different channel types: Unix domain sockets, TCP, UDP, and so on. The socket library provides specific classes for handling the common transports as well as a generic interface for handling the rest.

Sockets have their own vocabulary:

Term	Description
Domain	The family of protocols that is used as the transport mechanism. These values are constants such as AF_INET, PF_INET, PF_UNIX, PF_X25, and so on.
Type	The type of communications between the two endpoints, typically SOCK_STREAM for connection-oriented protocols and SOCK_DGRAM for connectionless protocols.
Protocol	Typically zero, this may be used to identify a variant of a protocol within a domain and type.
hostname	The identifier of a network interface: • A string, which can be a host name, a dotted-quad address, or an IPV6 address in colon (and possibly dot) notation • A string "<broadcast>", which specifies an INADDR_BROADCAST address. • A zero-length string, which specifies INADDR_ANY, or • An Integer, interpreted as a binary address in host byte order.
Port	Each server listens for clients calling on one or more ports. A port may be a Fixnum port number, a string containing a port number, or the name of a service.

Table 2.8.1 Description of socket terminology

The socket module:

To create a socket, you must use the socket. socket() function available in socket module, which has the general syntax −

s = socket.socket (socket_family, socket_type, protocol=0)

Here is the description of the parameters −

1. **socket_family:** This is either AF_UNIX or AF_INET, as explained earlier.
2. **socket_type:** This is either SOCK_STREAM or SOCK_DGRAM.
3. **protocol:** This is usually left out, defaulting to 0.

Once you have socket object, then you can use required functions to create your client or server program. Following is the list of functions required.

Server Socket Methods

Method	Description
s.bind()	This method binds address (hostname, port number pair) to socket.
s.listen()	This method sets up and start TCP listener.
s.accept()	This passively accept TCP client connection, waiting until connection arrives (blocking).

Table 2.8.2 Server socket methods

Client Socket Methods

Method	Description
s.connect()	This method actively initiates TCP server connection.

Table 2.8.3 Client socket methods

General Socket Methods

Method	Description
s.recv()	This method receives TCP message
s.send()	This method transmits TCP message
s.recvfrom()	This method receives UDP message
s.sendto()	This method transmits UDP message
s.close()	This method closes socket
socket.gethostname()	Returns the hostname.

Table 2.8.4 General socket methods

- **A Simple Server**

1. To write Internet servers, we use the **socket** function available in socket module to create a socket object. A socket object is then used to call other functions to setup a socket server.

2. Now call **bind(hostname, port)** function to specify a port for your service on the given host.

3. Next, call the accept method of the returned object. This method waits until a client connects to the port you specified, and then returns a connection object that represents the connection to that client.

```
#!/usr/bin/python  # This is server.py file

import socket  # Import socket module

s = socket.socket() # Create a socket object

host = socket.gethostname() # Get local machine name

port =12345 # Reserve a port for your service.
```

```
s.bind((host, port)) # Bind to the port

s.listen(5) # Now wait for client connection.whileTrue:

c, addr = s.accept() # Establish connection with client.

print'Got connection from', addr

c.send('Thank you for connecting')

c.close() # Close the connection
```

- **A Simple Client**

1. Let us write a very simple client program which opens a connection to a given port 12345 and given host. This is very simple to create a socket client using Python's socket module function.
2. The **socket.connect(host name, port)** opens a TCP connection to hostname on the port. Once you have a socket open, you can read from it like any IO object. When done, remember to close it, as you would close a file.

The following code is a very simple client that connects to a given host and port, reads any available data from the socket, and then exits –

```
#!/usr/bin/python  # This is client.py file

import socket   # Import socket module

s = socket.socket()  # Create a socket object

host = socket.gethostname() # Get local machine name

port =12345 # Reserve a port for your service.

s.connect((host, port))

print s.recv(1024)

s.close                    # Close the socket when done
```

Now run this server.py in background and then run above client.py to see the result.

```
# Following would start a server in background.

$ python server.py &

# Once server is started run client as follows:

$ python client.py
```

This would produce following result −

```
Got connection from('127.0.0.1',48437)Thank you for connecting
```

- **Python Internet modules**

A list of some important modules in Python Network/Internet programming.

Protocol	Common function	Port No	Python module
HTTP	Web pages	80	httplib, urllib, xmlrpclib
NNTP	Usenet news	119	Nntplib
FTP	File transfers	20	ftplib, urllib
SMTP	Sending email	25	Smtplib
POP3	Fetching email	110	Poplib
IMAP4	Fetching email	143	Imaplib
Telnet	Command lines	23	Telnetlib
Gopher	Document transfers	70	gopherlib, urllib

Table 2.8.5 Python Internet modules

Please check all the libraries mentioned above to work with FTP, SMTP, POP, and IMAP protocols.

2.9 TCP PROTOCOL

The transmission Control Protocol (TCP) is one of the most important protocols of Internet Protocols suite. It is most widely used protocol for data transmission in communication network such as internet.

Features:

- TCP is reliable protocol. That is, the receiver always sends either positive or negative acknowledgment about the data packet to the sender, so that the sender always has bright clue about whether the data packet is reached the destination or it needs to resend it.
- TCP ensures that the data reaches intended destination in the same order it was sent.
- TCP is connection oriented. TCP requires that connection between two remote points be established before sending actual data.
- TCP provides error-checking and recovery mechanism.
- TCP provides end-to-end communication.
- TCP provides flow control and quality of service.
- TCP operates in Client/Server point-to-point mode.
- TCP provides full duplex server, i.e. it can perform roles of both receiver and sender.

2.10 UDP PROTOCOL

The User Datagram Protocol (UDP) is simplest Transport Layer communication protocol available of the TCP/IP protocol suite. It involves minimum amount of communication mechanism. UDP is said to be an unreliable transport protocol but it uses IP services which provides best effort delivery mechanism.

In UDP, the receiver does not generate an acknowledgment of packet received and in turn, the sender does not wait for any acknowledgment of packet sent. This shortcoming makes this protocol unreliable as well as easier on processing.

Requirement of UDP:

A question may arise, why do we need an unreliable protocol to transport the data? We deploy UDP where the acknowledgment packets share significant amount of bandwidth along with the actual data. For example, in case of video streaming, thousands of packets are forwarded towards its users. Acknowledging all the packets is troublesome and may contain huge amount of bandwidth wastage. The best delivery mechanism of underlying IP protocol ensures best efforts to deliver its packets, but even if some packets in video streaming get lost, the impact is not calamitous and can be ignored easily. Loss of few packets in video and voice traffic sometimes goes unnoticed.

Features:

- UDP is used when acknowledgment of data does not hold any significance.
- UDP is good protocol for data flowing in one direction.
- UDP is simple and suitable for query based communications.
- UDP is not connection oriented.
- UDP does not provide congestion control mechanism.
- UDP does not guarantee ordered delivery of data.
- UDP is stateless.
- UDP is suitable protocol for streaming applications such as VoIP, multimedia streaming.

2.11 DIFFERENCES BETWEEN TCP AND UDP

	TCP	**UDP**
Acronym	Transmission Control Protocol	User/Universal Datagram Protocol
Connection	TCP is a connection-oriented protocol.	UDP is a connectionless protocol.
Function	As a message makes its way across the internet from one computer to another. This is connection based.	This protocol used in message transfer. Here one program can send a load of packets to another and that would be the end of the relationship.

Usage	TCP is suited for applications that require high reliability, and transmission time is relatively less critical.	UDP is suitable for applications that need fast, efficient transmission, such as games. UDP's stateless nature is also useful for servers that answer small queries from huge numbers of clients.
Use by otherprotocols	HTTP, HTTPs, FTP, SMTP, Telnet	DNS, DHCP, TFTP, SNMP, RIP, VOIP.
Ordering of data packets	TCP rearranges data packets in the order specified.	UDP has no inherent order as all packets are independent of each other. If ordering is required, it has to be managed by the application layer.
Speed of transfer	The speed for TCP is slower than UDP.	UDP is faster because error recovery is not attempted. It is a "best effort" protocol.
Reliability	There is absolute guarantee that the data transferred remains intact and arrives in the same order in which it was sent.	There is no guarantee that the messages or packets sent would reach at all.
Header Size	TCP header size is 20 bytes	UDP Header size is 8 bytes.
Common Header Fields	Source port, Destination port, Check Sum	Source port, Destination port, Check Sum
Streaming of data	Data is read as a byte stream, no distinguishing indications are transmitted to signal message (segment) boundaries.	Packets have definite boundaries which are honored upon receipt, meaning a read operation at the receiver socket will yield an entire message as it was originally sent.

Weight	TCP is heavy-weight. TCP requires three packets to set up a socket connection, before any user data can be sent. TCP handles reliability and congestion control.	UDP is lightweight. There is no ordering of messages, no tracking connections, etc. It is a small transport layer designed on top of IP.
Data Flow Control	TCP does Flow Control. TCP requires three packets to set up a socket connection, before any user data can be sent. TCP handles reliability and congestion control.	UDP does not have an option for flow control
Error Checking	TCP does error checking and error recovery. Erroneous packets are retransmitted from the source to the destination.	UDP does error checking but simply discards erroneous packets. Error recovery is not attempted.
Fields	1. Sequence Number, 2. AcK number, 3. Data offset, 4. Reserved, 5. Control bit, 6. Window, 7. Urgent Pointer 8. Options, 9. Padding, 10. Check Sum, 11. Source port, 12. Destination port	1. Length, 2. Source port, 3. Destination port, 4. Check Sum
Acknowledgment	Acknowledgment segments	No Acknowledgment
Handshake	SYN, SYN-ACK, ACK	No handshake (connectionless)

Table 2.11 Differences between TCP and UDP

2.11.1 Differences in Data Transfer Features

TCP ensures a reliable and ordered delivery of a stream of bytes from user to server or vice versa. **UDP** is not dedicated to end to end connections and communication does not check readiness of receiver.

- **Reliability:**

  TCP is more reliable since it manages message acknowledgment and re-transmissions in case of lost parts. Thus there is absolutely no missing data. **UDP** does not ensure that communication has reached receiver since concepts of acknowledgment, time out and re-transmission are not present.

- **Ordering:**

  TCP transmissions are sent in a sequence and they are received in the same sequence. In the event of data segments arriving in wrong order, TCP reorders and delivers application. In the case of **UDP**, sent message sequence may not be maintained when it reaches receiving application. There is absolutely no way of predicting the order in which message will be received.

- **Connection:**

  TCP is a heavy weight connection requiring three packets for a socket connection and handles congestion control and reliability. **UDP** is a lightweight transport layer designed atop an IP. There are no tracking connections or ordering of messages.

- **Method of transfer:**

  TCP reads data as a byte stream and message is transmitted to segment boundaries. **UDP** messages are packets which are sent individually and on arrival are checked for their integrity. Packets have defined boundaries while data stream has none.

- **Error Detection:**

  UDP works on a "best-effort" basis. The protocol supports error detection via checksum but when an error is detected, the packet is discarded.

Re-transmission of the packet for recovery from that error is not attempted. This is because UDP is usually for time-sensitive applications like gaming or voice transmission. Recovery from the error would be pointless because by the time the re-transmitted packet is received, it won't be of any use.

TCP uses both error detection and error recovery. Errors are detected via checksum and if a packet is erroneous, it is not acknowledged by the receiver, which triggers a re-transmission by the sender. This operating mechanism is called Positive Acknowledgment with Re-transmission (PAR).

2.12 TIME FUNCTION IN PYTHON

The method **time()** returns the time as a floating point number expressed in seconds since the epoch, in UTC. Even though the time is always returned as a floating point number, not all systems provide time with a better precision than 1 second. While this function normally returns non-decreasing values, it can return a lower value than a previous call if the system clock has been set back between the two calls.

Syntax:

Following is the syntax for **time()** method:

```
time. time()
```

Return Value:

This method returns the time as a floating point number expressed in seconds since the epoch, in UTC.

Example:

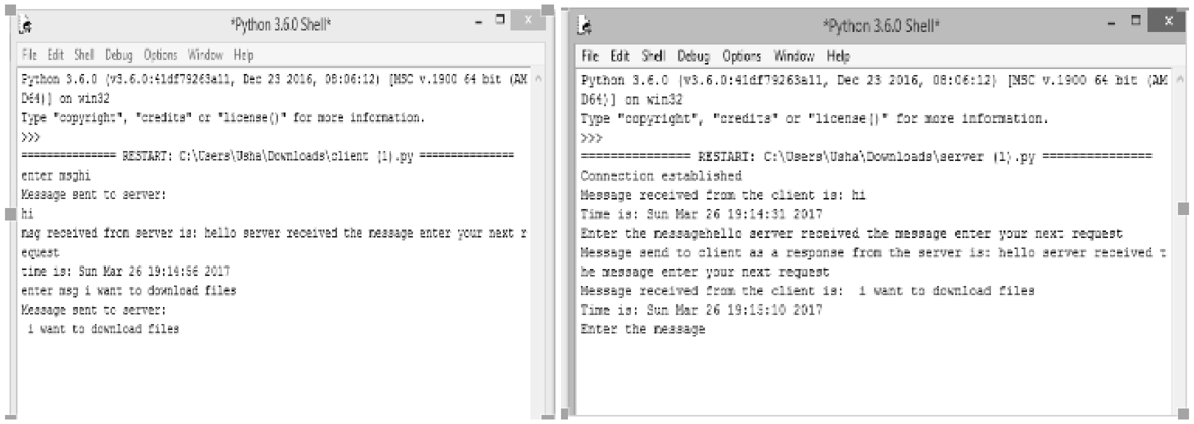
The following example shows the usage of time() method.

```
#!/usr/bin/python

import time

print"time.time():%f"%time.time()

print time.localtime(time.time())

print time.asctime(time.localtime(time.time()))
```

When we run above program, it produces following result:

time.time():1234892919.655932(2009,2,17,10,48,39,1,48,0)TueFeb1710:48:392009

SCREENSHOT OF CLIENT SERVER COMMUNICATION IN PYTHON



```
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 08:06:12) [MSC v.1900 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
=============== RESTART: C:\Users\Usha\Downloads\client (1).py ===============
enter msghi
Message sent to server:
hi
msg received from server is: hello server received the message enter your next r
equest
time is: Sun Mar 26 19:14:56 2017
enter msg i want to download files
Message sent to server:
 i want to download files
```

```
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 08:06:12) [MSC v.1900 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
=============== RESTART: C:\Users\Usha\Downloads\server (1).py ===============
Connection established
Message received from the client is: hi
Time is: Sun Mar 26 19:14:31 2017
Enter the messagehello server received the message enter your next request
Message send to client as a response from the server is: hello server received t
he message enter your next request
Message received from the client is:  i want to download files
Time is: Sun Mar 26 19:15:10 2017
Enter the message
```

Fig 2.12 Client Server communication

CHAPTER-3
DOCKER

3.1 INTRODUCTION TO DOCKER

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

Docker allows you to package an application with all of its dependencies into a complete file system that contains everything it needs to run: code, runtime, system tools, system libraries – anything you can install on a server. This guarantees that it will always run the same, regardless of the environment it is running in. Released in 2013.

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. Because of the lightweight nature of containers, which run without the extra load of a hypervisor, you can run more containers on a given hardware combination than if you were using virtual machines.

Docker provides tooling and a platform to manage the lifecycle of your containers:

- Encapsulate your applications (and supporting components) into Docker containers.
- Distribute and ship those containers to your teams for further development and testing.
- Deploy those applications to your production environment, whether it is in a local data center or the Cloud.

3.1.1 What is 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.

- 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.

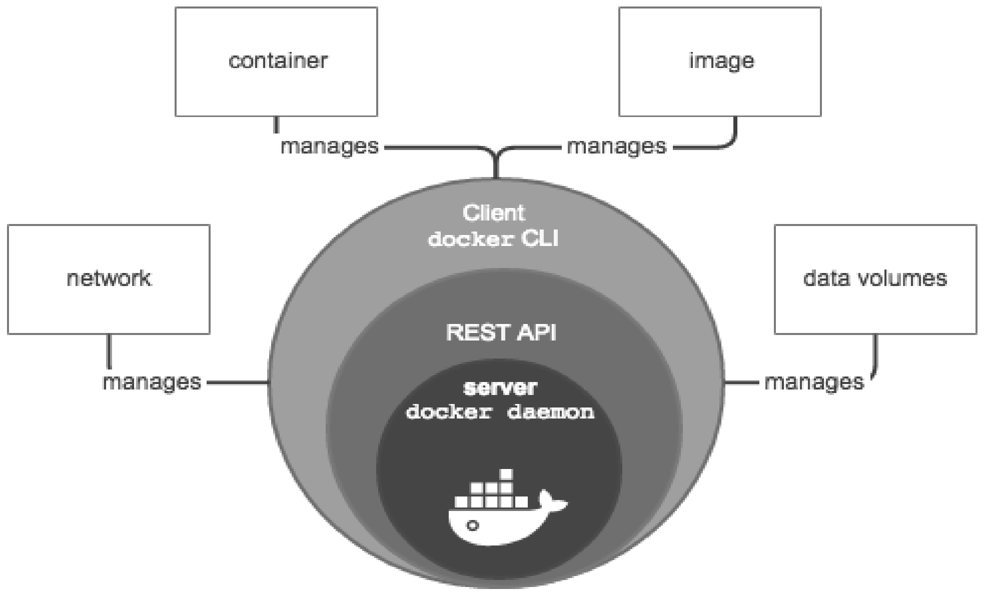


Fig 3.1.1 Docker Functionality

The CLI uses the Docker REST API to control or interact with the Docker daemon through scripting or direct CLI commands. Many other Docker applications use the underlying API and CLI. The daemon creates and manages Docker objects, such as images, containers, networks, and data volumes.

3.1.2 What can I use Docker for?

- **Fast, consistent delivery of your applications**

Docker can streamline the development lifecycle by allowing developers to work in standardized environments using local containers which provide your applications and services. You can also integrate Docker into your continuous integration and continuous deployment (CI/CD) workflow.

Consider the following example scenario. Your developers write code locally and share their work with their colleagues using Docker containers. They can use Docker to push their applications into a test environment and execute automated and manual tests. When developers find problems, they can fix them in the development environment and redeploy them to the test environment for testing. When testing is complete, getting the fix to the customer is as simple as pushing the updated image to the production environment.

- **Responsive deployment and scaling**

  Docker's container-based platform allows for highly portable workloads. Docker containers can run on a developer's local host, on physical or virtual machines in a data center, in the Cloud, or in a mixture of environments.

  Docker's portability and lightweight nature also make it easy to dynamically manage workloads, scaling up or tearing down applications and services as business needs dictate, in near real time.

- **Running more workloads on the same hardware**

  Docker is lightweight and fast. It provides a viable, cost-effective alternative to hypervisor-based virtual machines, allowing you to use more of your compute capacity to achieve your business goals. This is useful in high density environments and for small and medium deployments where you need to do more with fewer resources.

3.2 VIRTUALIZATION

Virtualization is the creation of a virtual -- rather than actual -- version of something, such as an operating system, a server, a storage device or network resources.

How virtualization works:

Virtualization describes a technology in which an application, guest operating system or data storage is abstracted away from the true underlying hardware or software. A key use of virtualization technology is server virtualization, which uses a software layer called a hypervisor to emulate the underlying hardware. This often includes the CPU's memory, I/O and network traffic. The guest operating system, normally interacting with true hardware, is now doing so with a software emulation of that hardware, and often the guest operating system has no idea it's on virtualized hardware. While the performance of this virtual system is not equal to the performance of the operating system running on true hardware, the concept of virtualization works because most guest operating systems and applications don't need the full use of the underlying hardware. This allows for greater flexibility, control and isolation by removing the dependency on a given hardware platform. While initially meant for server virtualization, the concept of virtualization has spread to applications, networks, data and desktops.

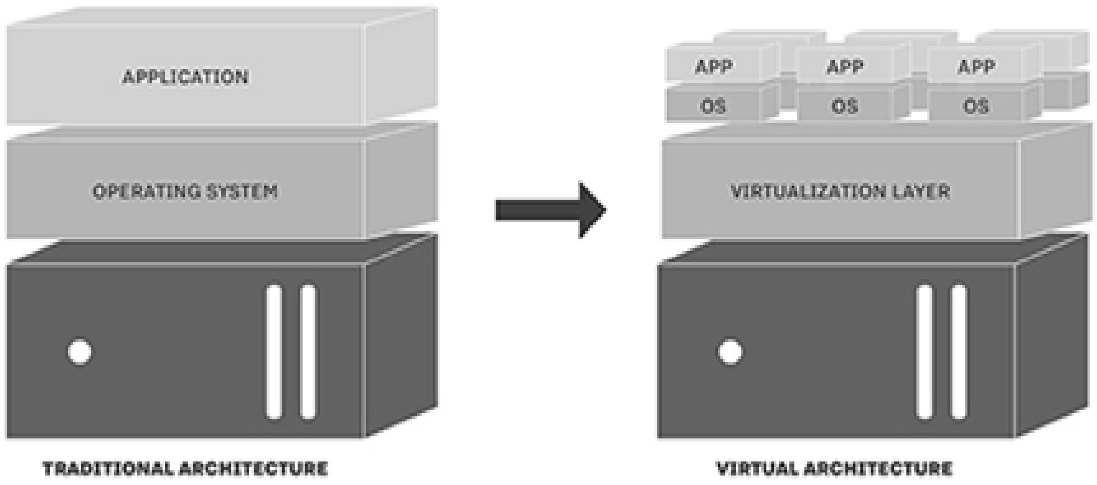


Fig 3.2 Traditional and Virtual Architecture

3.3 DISADVANTAGES OF VMWARE

- When multiple virtual machines are simultaneously running on a host computer, each virtual machine may introduce an unstable performance, which depends on the workload on the system by other running virtual machines;
- Virtual machine is not that efficient as a real one when accessing the hardware.

3.4 WHY DOCKER

Docker is used because it supports virtualization concept which makes the server respond in less amount of time.

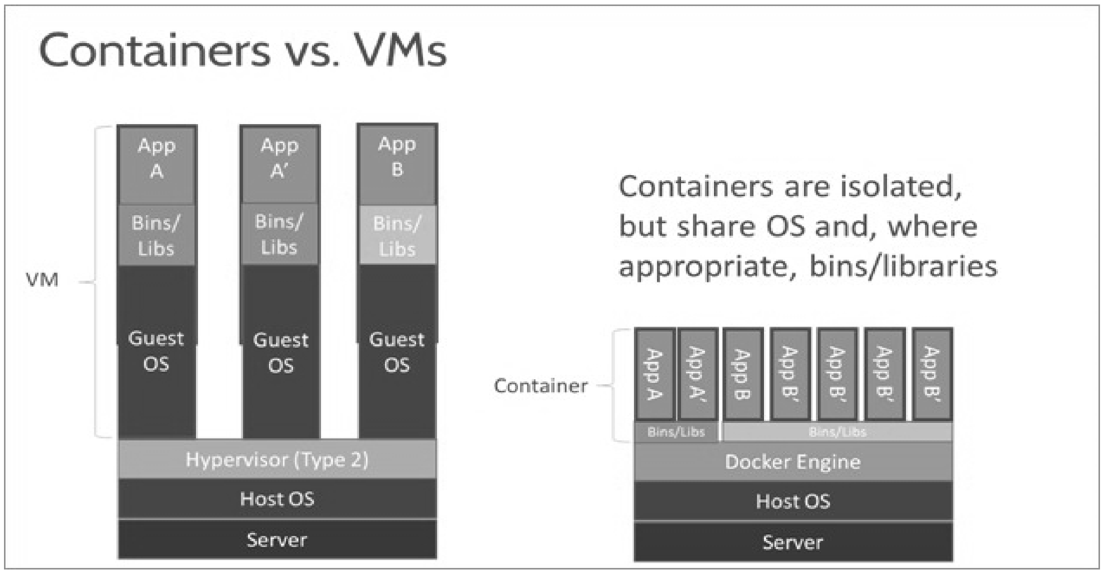


Fig 3.4 Containers Vs VMs

1. Simplifying Configuration

The primary use case Docker advocates is simplifying configuration. One of the big advantages of VM 's is the ability to run any platform with its own configuration on top of our infrastructure. Docker provides this same capability without the overhead of a virtual machine. It lets you put your environment and configuration into code and deploy it. The same Docker configuration can also be used in a variety of environments. This decouples infrastructure requirements from the application environment.

The freedom to run your applications across multiple IaaS/PaaS without any extra tweaks is the ultimate dream that Docker can help you achieve. Today, every IaaS/PaaS provider from Amazon to Google supports Docker. The big names have placed their bet on Docker. Now, it's your turn to benefit from these same options.

2. Code Pipeline Management

The previous use case makes a large impact in managing the code pipeline. As the code travels from the developer's machine to production, there are many different environments it has to go through to get there. Each of these may have minor differences along the way. Docker provides a consistent environment for the application from dev through production, easing the code development and deployment pipeline. The immutable nature of Docker images, and the ease with which they can be spun up, help you achieve zero change in application runtime environments across dev through production.

3. Developer Productivity

In a developer environment, we have two goals that are at odds with each other:

- We want it be as close as possible to production; and
- We want the development environment to be as fast as possible for interactive use.

Ideally, to achieve the first goal, we need to have every service running on its own VM to reflect how the production application runs. However, we don't want to always require an Internet connection and add the overhead of working remotely every time a compilation is needed.

This is where the low overhead of Docker comes in handy. A development environment usually has a low memory capacity, and by not adding to the memory footprint that's commonly done when using a VM, Docker easily allows a few dozen services to run.

To achieve the second goal, to provide a fast feedback loop, we use Docker's shared volumes to make the application code available to the container(s) from the container's host OS, which is a virtual box VM (typically, a Vagrant box). The application source code is made available to the container host OS (Vagrant box) using Vagrant's synced folders with the host OS (Windows, Mac or Linux).

This approach has multiple benefits. The developer can edit the source code from his platform of choice (Windows, Mac or Linux) and is able to see the changes right away as the applications run using the same source code with the running environment set inside of the Vagrant box using Docker container(s).

Moreover, this approach helps a front-end engineer who is not much into the back end nitty gritty to easily use the full application setup and work on his or her area of interest without the setup or installation blues getting in the way. And, it provides an optional opportunity for further exploration on how back-end systems work under the hood to get a better understanding for the full stack.

4. App Isolation

There may be many reasons for which you end up running multiple applications on the same machine. An example of this is the developer productivity flow described earlier. But there are other cases, too. A couple of such cases to consider are server consolidation for decreasing cost or a gradual plan to separate a monolithic application into decoupled pieces. Let's say, for example, you need to run two REST API servers, both of which use flask. But, each of them uses a slightly different version of flask and other such dependencies. Running these API servers under different containers provides an easy way out through what we call the "dependency hell."

5. Server Consolidation

Just like using VMs for consolidating multiple applications, the application isolation abilities of Docker allows consolidating multiple servers to save on cost. However, without the memory footprint of multiple OSes and the ability to share unused memory across the instances, Docker provides far denser server consolidation than you can get with VMs.The new breed of highly customizable PAAS, such as Heroku, Elastic Beanstalk and App Engine, all use these powerful feature of containers that is now at your disposal with Docker. Moreover, open source projects like Deis, Kubernetes, Cadvisor, Panamax, and others make deploying and monitoring large numbers of containers representing a multi-tier application architecture manageable.

6. Debugging Capabilities

Docker provides many tools that are not necessarily specific to containers, but, they work well with the concept of containers. They also provide extremely useful functionality. This includes the ability to checkpoint containers and container versions, as well as to diff two containers. This can be immensely useful in fixing an application.

Flux7.com was run inside a Docker container. Our web developer told us that a crash resulted from a code change he'd pushed from the UI to the functions.php file. Within a minute, I was able to create a dev environment enabling the web developer to debug in a sandbox. When he gave us the green light a short time later, we were able to switch back to the latest version of the website, thanks to Docker and Linux Containers.

While the process was solvable using another strategy, using Docker was an efficient way to solve the problem. And, it's one we have implemented in a number of customer deployments where front-end functionality is business-critical.

7. Multi-tenancy

Yet another interesting use case of Docker is its use in multi-tenant applications, thereby avoiding major application rewrites. Our own example is to develop quick and easy multi-tenancy for an IoT application. Code bases for such multi-tenant applications are far more complicated, rigid and difficult to handle. Re architecting an application is not only time consuming, but also costs a lot of money.

Using Docker, it was easy and inexpensive to create isolated environments for running multiple instances of app tiers for each tenant. This was possible given the spin up speed of Docker environments and it's easy-to-use API, which we can use to spin containers programmatically. We used docker-py, which is a Python library to help interact with the Docker daemon through a web application interface.

8. Rapid Deployment

Before VMs, bringing up a new hardware resource took days. Virtualization brought this number down to minutes. Docker, by creating just a container for the process and not booting up an OS, brings it down to seconds. This is the enabling technology that has brought Google and Facebook to using containers.

Essentially, you can create and destroy resources in your data center without worrying about the cost of bringing it up again. With typical data center utilization at 30%, it is easy to bump up that number by using a more aggressive allocation of resources. And, the low cost of bringing up a new instance allows for a more aggressive allocation of resources.

Moreover, the immutable nature of Docker images gives you the peace of mind that things will work exactly the way they have been working and are supposed to work.

3.5 WHAT IS DOCKER ARCHITECTURE?

Docker uses a client-server architecture. The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface.

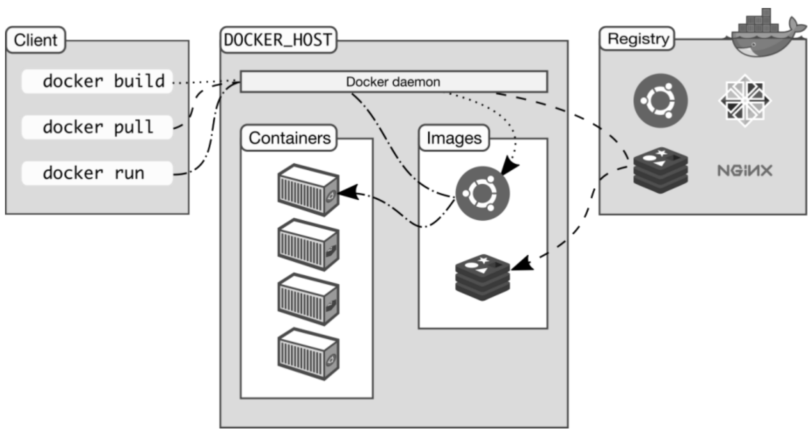


Fig 3.5 Docker architecture

The Docker daemon: The Docker daemon runs on a host machine. The user uses the Docker client to interact with the daemon.

The Docker client: The Docker client, in the form of the docker binary, is the primary user interface to Docker. It accepts commands and configuration flags from the user and communicates with a Docker daemon. One client can even communicate with multiple unrelated daemons.

Inside Docker: To understand Docker's internals, you need to know about images, registries, and containers.

DOCKER IMAGES

A Docker image is a read-only template with instructions for creating a Docker container. For example, an image might contain an Ubuntu operating system with Apache web server and your web application installed. You can build or update images from scratch or download and use images created by others. An image may be based on, or may extend, one or more other images. A docker image is described in text file called a Dockerfile, which has a simple, well-defined syntax. Docker images are the build component of Docker.

DOCKER CONTAINERS

A Docker container is a runnable instance of a Docker image. You can run, start, stop, move, or delete a container using Docker API or CLI commands. When you run a container, you can provide configuration metadata such as networking information or environment variables. Each container is an isolated and secure application platform, but can be given access to resources running in a different host or container, as well as persistent storage or databases. Docker containers are the run component of Docker.

DOCKER REGISTRIES

A docker registry is a library of images. A registry can be public or private, and can be on the same server as the Docker daemon or Docker client, or on a totally separate server. Docker registries are the distribution component of Docker.

DOCKER SERVICES

A Docker service allows a swarm of Docker nodes to work together, running a defined number of instances of a replica task, which is itself a Docker image. You can specify the number of concurrent replica tasks to run, and the swarm manager ensures that the load is spread evenly across the 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. Docker services are the scalability component of Docker.

3.6 ADVANTAGES OF USING DOCKER

Docker brings in an API for container management, an image format and a possibility to use a remote registry for sharing containers. This scheme benefits both developers and system administrators with advantages such as:

- **Rapid application deployment** – containers include the minimal runtime requirements of the application, reducing their size and allowing them to be deployed quickly.

- **Portability across machines** – an application and all its dependencies can be bundled into a single container that is independent from the host version of Linux kernel, platform distribution, or deployment model. This container can be transfered to another machine that runs Docker, and executed there without compatibility issues.

- **Version control and component reuse** – you can track successive versions of a container, inspect differences, or roll-back to previous versions. Containers reuse components from the preceding layers, which makes them noticeably lightweight.

- **Sharing** – you can use a remote repository to share your container with others. Red Hat provides a registry for this purpose, and it is also possible to configure your own private repository.

- **Lightweight footprint and minimal overhead** – Docker images are typically very small, which facilitates rapid delivery and reduces the time to deploy new application containers.

- **Simplified maintenance** – Docker reduces effort and risk of problems with application dependencies.

3.7 REAL TIME APPLICATIONS OF DOCKER

Ebay simplifies application deployment:

- **Background**

  Ebay is on a mission to be the world's favorite destination for discovering great value and unique selection. The company provides a platform for sellers to receive the solutions, and support they need to grow their businesses and thrive**.**

- **Challenge**

  When the company was first launching eBay Now, there same day delivery service, they started off attempting to use VMs to help optimize their app development process. They discovered that there were limitations of VMs.

- **Solutions**

  eBay uses Docker for their continuous integration process. They leverage Docker containers to implement an efficient, automated path from the developer's laptop through test and QA. Now all containerized applications go directly into CI, while they run database driven tests in parallel. Using Docker now they have a process that works on individual developer laptops, communal resources and in production.

Uber accelerates developer on boarding from weeks to minutes using docker:

- **Background**

  Uber is evolving the way the world moves. By seamlessly connecting riders to drivers through their apps, Uber makes cities more accessible, and more business.

- **Challenge**

  Uber's communication tool that they use to manage their services provisioning was struggling to keep up with the fast pace of the company. At small scale the tool worked fine for managing but as the company experienced rapid growth, the tool began to falter. At one point, it was taking months for apps to be onboard into cluster.

- **Solution**

  Docker provides consistency for both build and run time environments. It has helped Uber reduce their footprint of Debian packages as well. Makes it easy to updates certain images without having to reboot then entire fleet. They are now onboarding all of there new services into Docker, and will be onboarding all of their existing applications into Docker clusters. Docker containers also reduce time from weeks and months to minutes or hours, providing an isolation of resources so that applications no longer interference with one another.

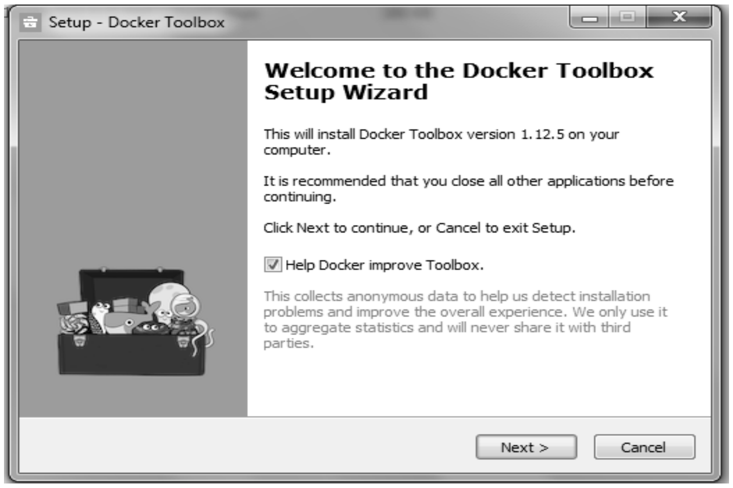
Following is non exhaustive list of companies using Docker.

1. The New York Times
2. PayPal
3. Business Insider
4. Cornell University (Not a company but still can be considered)
5. Splunk
6. The Washington Post
7. Swisscomm
8. GE
9. Groupon
10. Yandex
11. Uber

12. Ebay
13. Shopify
14. Spotify
15. New Relic
16. Yelp

3.8 INSTALLATION OF DOCKER

Step 1: Press on Next


Setup - Docker Toolbox

Welcome to the Docker Toolbox Setup Wizard

This will install Docker Toolbox version 1.12.5 on your computer.

It is recommended that you close all other applications before continuing.

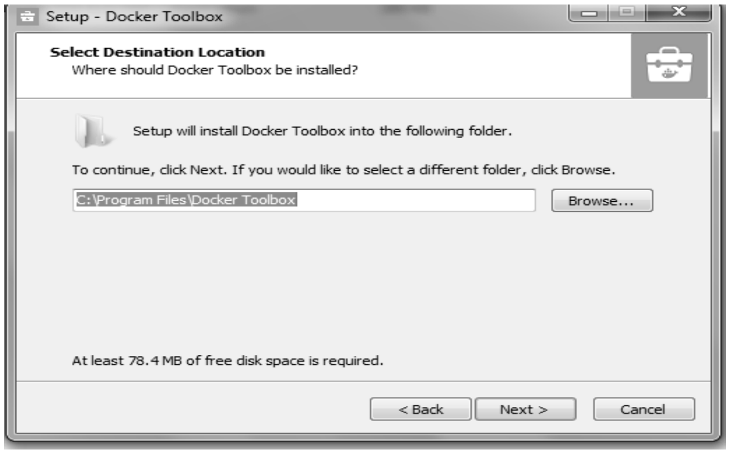
Click Next to continue, or Cancel to exit Setup.

☑ Help Docker improve Toolbox.

This collects anonymous data to help us detect installation problems and improve the overall experience. We only use it to aggregate statistics and will never share it with third parties.

Fig 3.8.1 Docker Installation step 1.

Step 2: After browsing the required folder for the program files press on Next.


Setup - Docker Toolbox

Select Destination Location
Where should Docker Toolbox be installed?

Setup will install Docker Toolbox into the following folder.

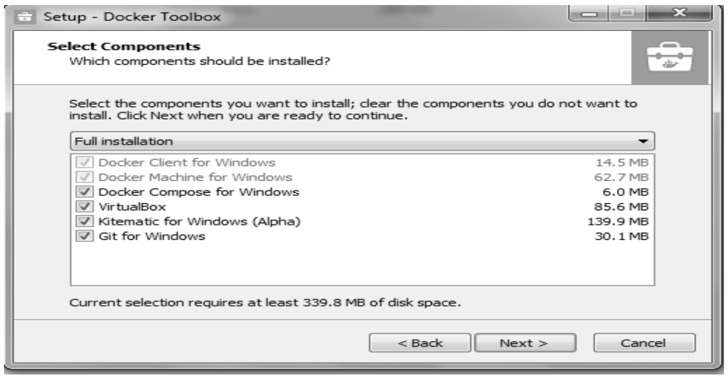
To continue, click Next. If you would like to select a different folder, click Browse.

C:\Program Files\Docker Toolbox

At least 78.4 MB of free disk space is required.

Fig 3.8.2 Docker Installation step 2.

Step 3: After selecting the components press on Next.


Setup - Docker Toolbox

Select Components
Which components should be installed?

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

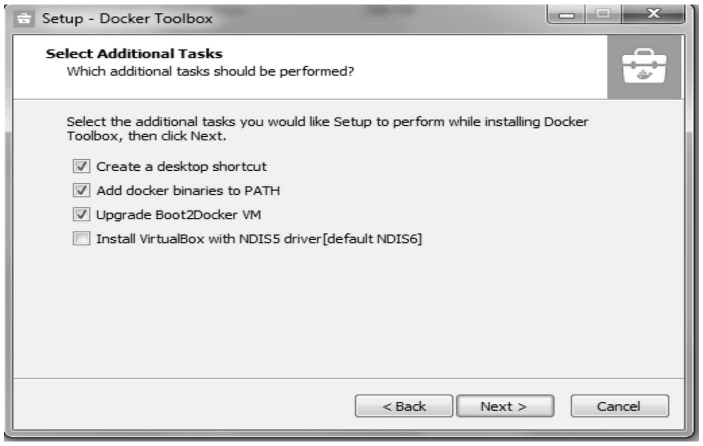
Full installation

☑ Docker Client for Windows	14.5 MB
☑ Docker Machine for Windows	62.7 MB
☑ Docker Compose for Windows	6.0 MB
☑ VirtualBox	85.6 MB
☑ Kitematic for Windows (Alpha)	139.9 MB
☑ Git for Windows	30.1 MB

Current selection requires at least 339.8 MB of disk space.

Fig 3.8.3 Docker Installation step 3.

Step 4: After selecting Additional tasks press on Next


Setup - Docker Toolbox

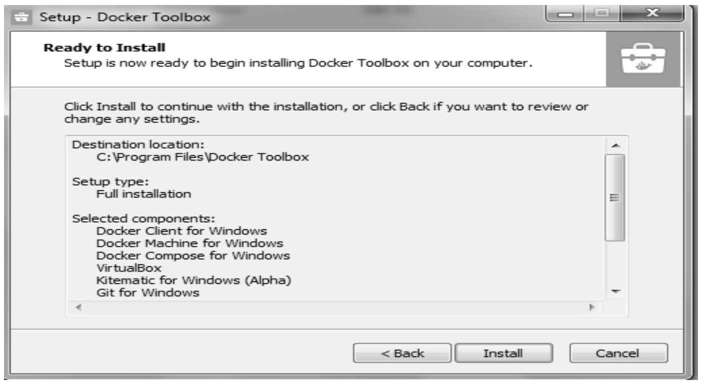
Select Additional Tasks
Which additional tasks should be performed?

Select the additional tasks you would like Setup to perform while installing Docker Toolbox, then click Next.

☑ Create a desktop shortcut
☑ Add docker binaries to PATH
☑ Upgrade Boot2Docker VM
☐ Install VirtualBox with NDIS5 driver[default NDIS6]

Fig 3.8.4 Docker Installation step4

Step 5: Now press on Install Button


Setup - Docker Toolbox

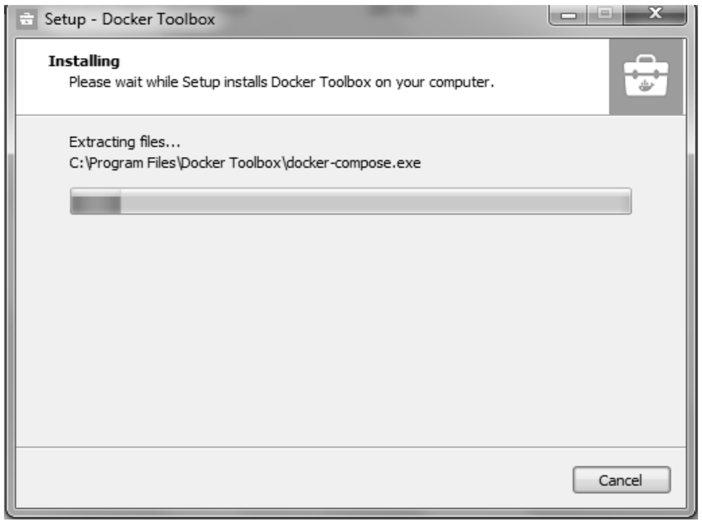
Ready to Install
Setup is now ready to begin installing Docker Toolbox on your computer.

Click Install to continue with the installation, or click Back if you want to review or change any settings.

```
Destination location:
      C:\Program Files\Docker Toolbox

Setup type:
      Full installation

Selected components:
      Docker Client for Windows
      Docker Machine for Windows
      Docker Compose for Windows
      VirtualBox
      Kitematic for Windows (Alpha)
      Git for Windows
```

Fig 3.8.5 Docker Installation step 5.

Step 6: The setup is Installing.

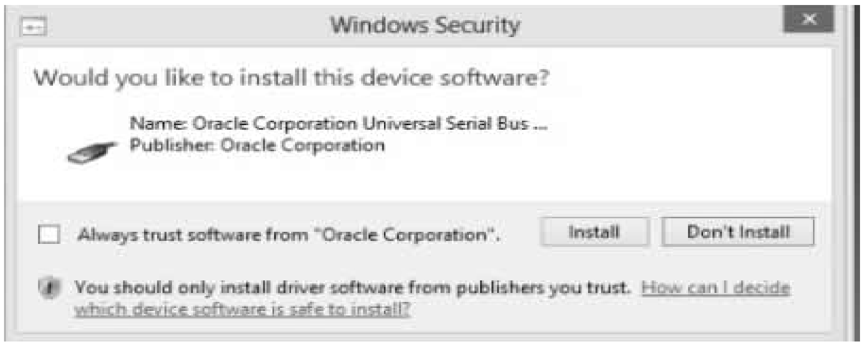

Setup - Docker Toolbox

Installing
Please wait while Setup installs Docker Toolbox on your computer.

Extracting files...
C:\Program Files\Docker Toolbox\docker-compose.exe

Fig 3.8.6 Docker Installation step 6.

Steps 7-9: press on install buttons in further steps.


Windows Security

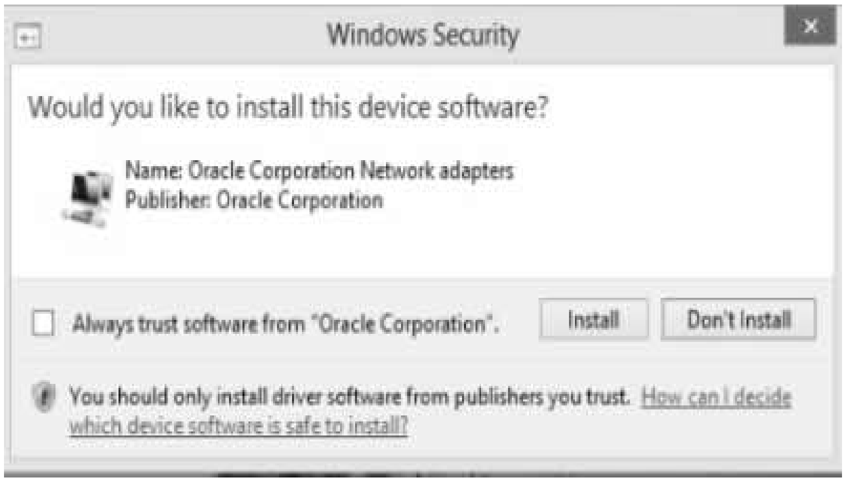
Would you like to install this device software?

Name: Oracle Corporation Universal Serial Bus ...
Publisher: Oracle Corporation

Always trust software from "Oracle Corporation".

You should only install driver software from publishers you trust. How can I decide which device software is safe to install?

Fig 3.8.7 Docker Installation step 7.


Windows Security

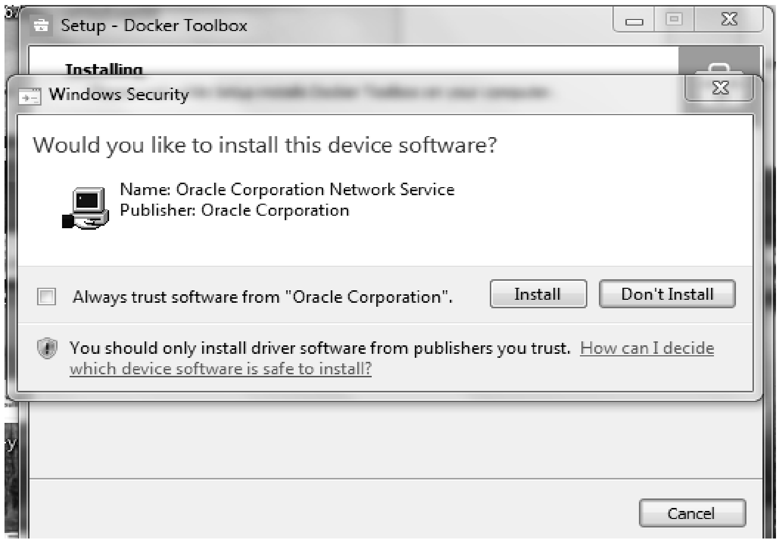
Would you like to install this device software?

Name: Oracle Corporation Network adapters
Publisher: Oracle Corporation

Always trust software from "Oracle Corporation".

You should only install driver software from publishers you trust. How can I decide which device software is safe to install?

Fig 3.8.8 Docker Installation step 8.


Setup - Docker Toolbox

Installing

Windows Security

Would you like to install this device software?

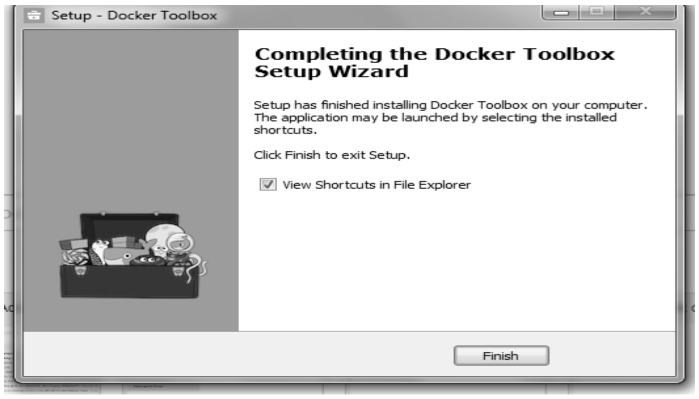
Name: Oracle Corporation Network Service
Publisher: Oracle Corporation

Always trust software from "Oracle Corporation".

You should only install driver software from publishers you trust. How can I decide which device software is safe to install?

Fig 3.8.9 Docker Installation step 9.

Step 10: Click on Finish.


Setup - Docker Toolbox

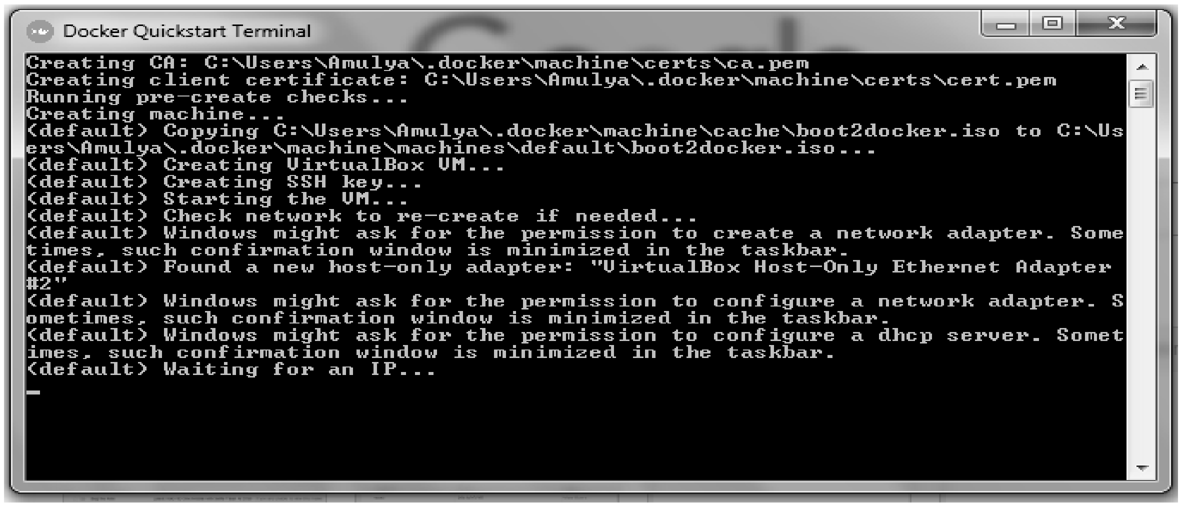
Completing the Docker Toolbox Setup Wizard

Setup has finished installing Docker Toolbox on your computer. The application may be launched by selecting the installed shortcuts.

Click Finish to exit Setup.

[✓] View Shortcuts in File Explorer

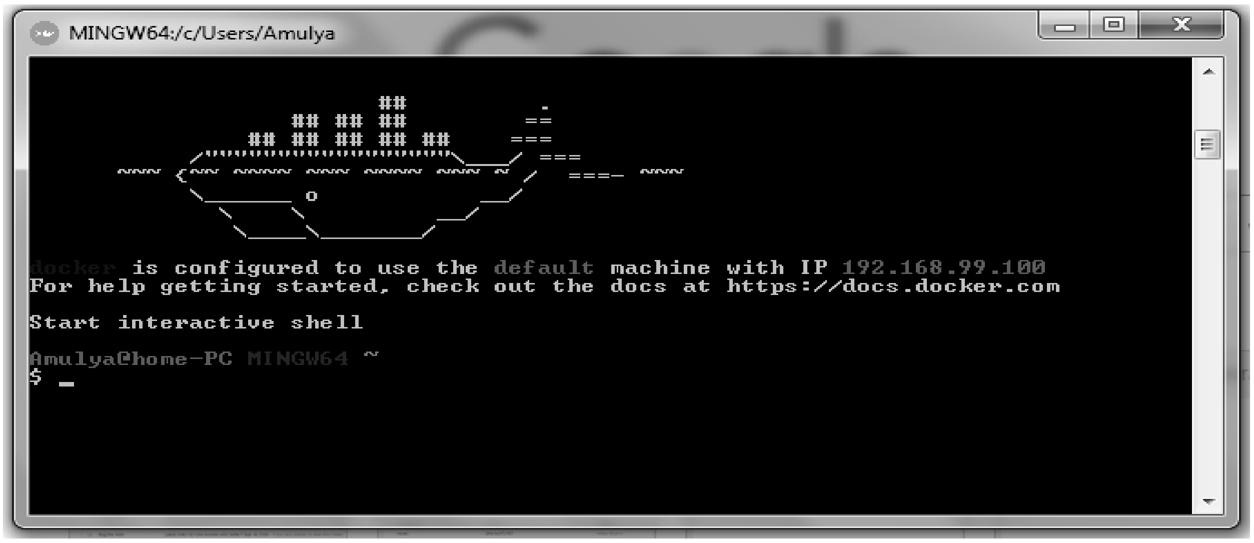
Fig 3.8.10 Docker Installation step 10.


Docker Quickstart Terminal

```
Creating CA: C:\Users\Amulya\.docker\machine\certs\ca.pem
Creating client certificate: C:\Users\Amulya\.docker\machine\certs\cert.pem
Running pre-create checks...
Creating machine...
(default) Copying C:\Users\Amulya\.docker\machine\cache\boot2docker.iso to C:\Us
ers\Amulya\.docker\machine\machines\default\boot2docker.iso...
(default) Creating VirtualBox VM...
(default) Creating SSH key...
(default) Starting the VM...
(default) Check network to re-create if needed...
(default) Windows might ask for the permission to create a network adapter. Some
times, such confirmation window is minimized in the taskbar.
(default) Found a new host-only adapter: "VirtualBox Host-Only Ethernet Adapter
#2"
(default) Windows might ask for the permission to configure a network adapter. S
ometimes, such confirmation window is minimized in the taskbar.
(default) Windows might ask for the permission to configure a dhcp server. Somet
imes, such confirmation window is minimized in the taskbar.
(default) Waiting for an IP...
_
```

Fig 3.8.11 Initialization of Docker Terminal

Docker console:The docker terminal is ready to use.



```
                        ##         .
                  ## ## ##        ==
               ## ## ## ## ##    ===
           /"""""""""""""""""\___/ ===
      ~~~ {~~ ~~~~ ~~~ ~~~~ ~~~ ~ /  ===- ~~~
           \______ o           __/
             \    \         __/
              \____\_______/

docker is configured to use the default machine with IP 192.168.99.100
For help getting started, check out the docs at https://docs.docker.com

Start interactive shell

Amulya@home-PC MINGW64 ~
$ _
```

Fig 3.8.12 Docker console

3.9 DOCKER COMMANDS

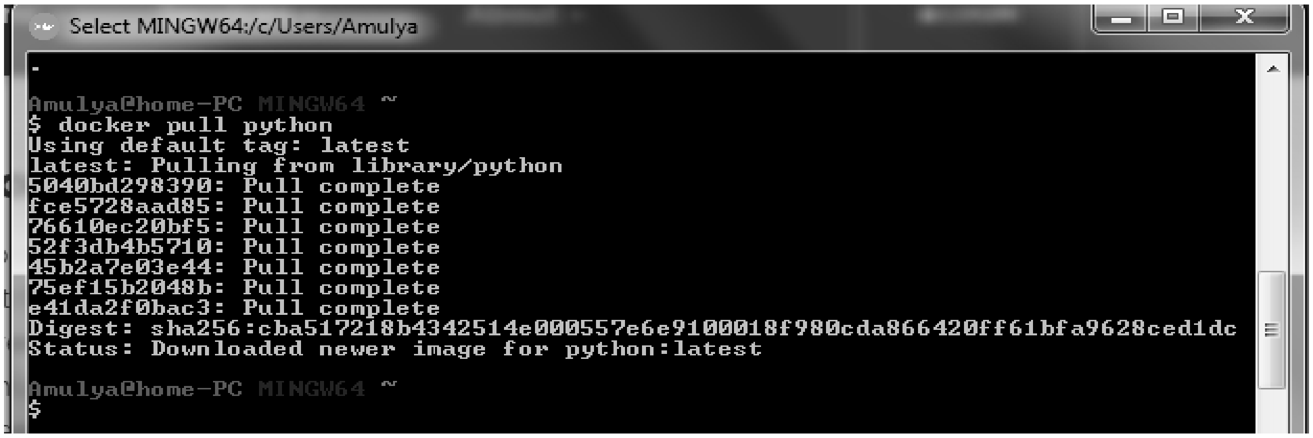
Docker commands are used for creating a docker file, pulling images, building images, running the images. Some of the commands are as follows:

➢ **Pull**: Pull an image or a repository from a registry

Syntax:

docker pull [OPTIONS] NAME[:TAG|@DIGEST]

Example:



```
-
Amulya@home-PC MINGW64 ~
$ docker pull python
Using default tag: latest
latest: Pulling from library/python
5040bd298390: Pull complete
fce5728aad85: Pull complete
76610ec20bf5: Pull complete
52f3db4b5710: Pull complete
45b2a7e03e44: Pull complete
75ef15b2048b: Pull complete
e41da2f0bac3: Pull complete
Digest: sha256:cba517218b4342514e000557e6e9100018f980cda866420ff61bfa9628ced1dc
Status: Downloaded newer image for python:latest

Amulya@home-PC MINGW64 ~
$
```

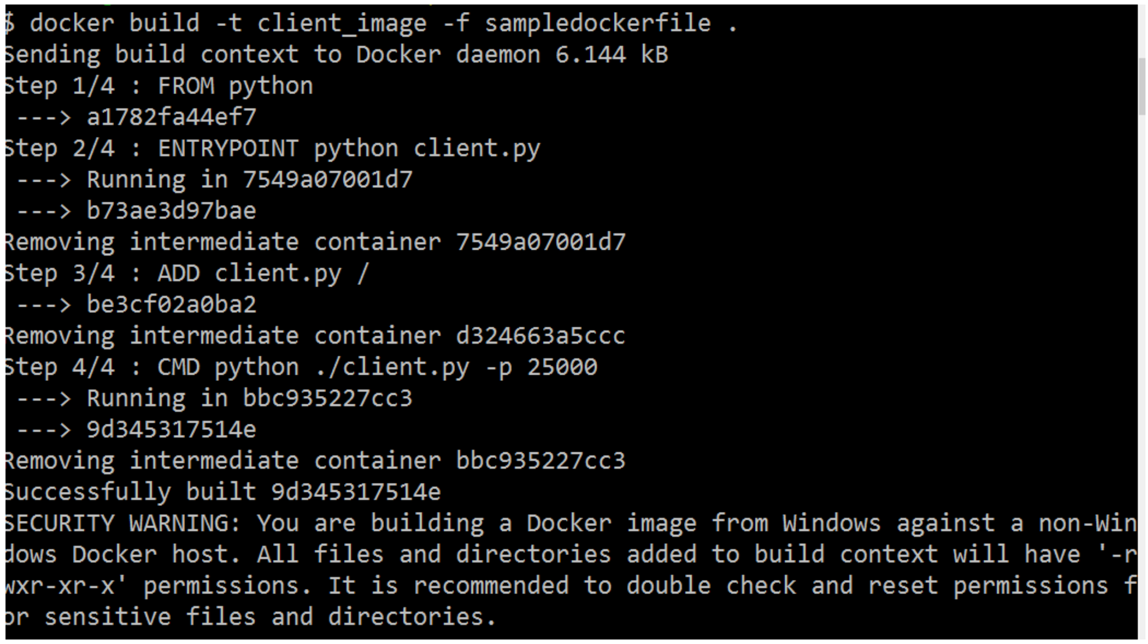
Fig 3.9.1   Pull command example

➢ **Build**: Build an image from a Dockerfile

Syntax:

docker build [OPTIONS]   PATH | URL | -

Example:



```
$ docker build -t client_image -f sampledockerfile .
Sending build context to Docker daemon 6.144 kB
Step 1/4 : FROM python
 ---> a1782fa44ef7
Step 2/4 : ENTRYPOINT python client.py
 ---> Running in 7549a07001d7
 ---> b73ae3d97bae
Removing intermediate container 7549a07001d7
Step 3/4 : ADD client.py /
 ---> be3cf02a0ba2
Removing intermediate container d324663a5ccc
Step 4/4 : CMD python ./client.py -p 25000
 ---> Running in bbc935227cc3
 ---> 9d345317514e
Removing intermediate container bbc935227cc3
Successfully built 9d345317514e
SECURITY WARNING: You are building a Docker image from Windows against a non-Win
dows Docker host. All files and directories added to build context will have '-r
wxr-xr-x' permissions. It is recommended to double check and reset permissions f
or sensitive files and directories.
```

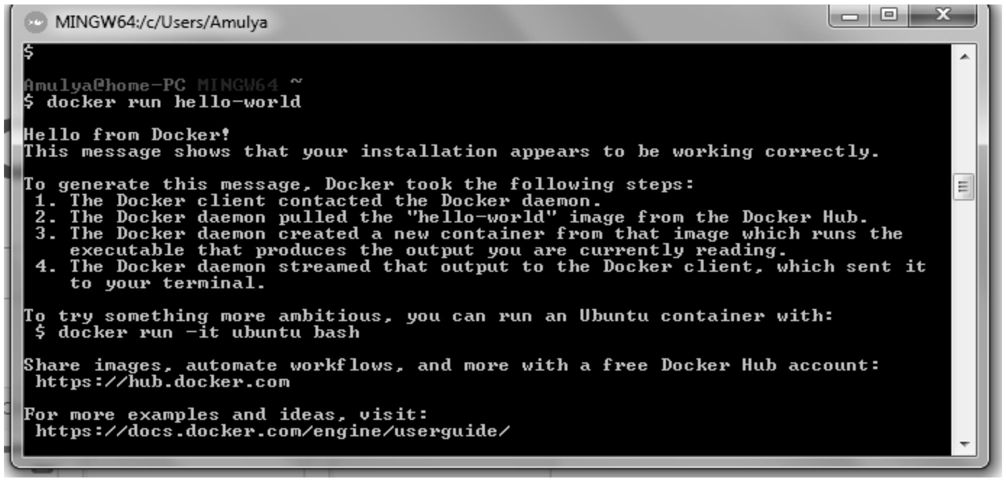
Fig 3.9.2   Build command example

➢ **Run:** Run a command in a new container

Syntax:

docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

Example:



```
$

Amulya@home-PC MINGW64 ~
$ 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.
 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 Hub account:
 https://hub.docker.com

For more examples and ideas, visit:
 https://docs.docker.com/engine/userguide/
```

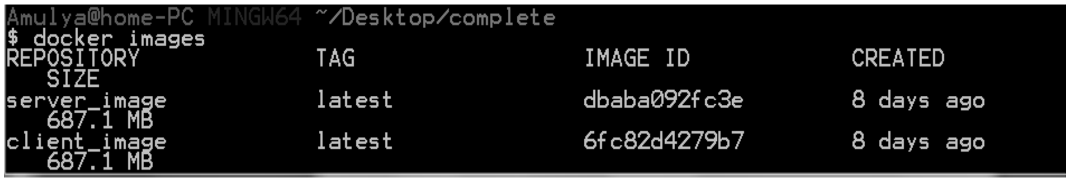
Fig 3.9.3   run command example

➢ **Images**: List images

Syntax:

docker images [OPTIONS] [REPOSITORY[:TAG]]

Example:



```
Amulya@home-PC MINGW64 ~/Desktop/complete
$ docker images
REPOSITORY              TAG                 IMAGE ID            CREATED
   SIZE
server_image            latest              dbaba092fc3e        8 days ago
   687.1 MB
client_image            latest              6fc82d4279b7        8 days ago
   687.1 MB
```

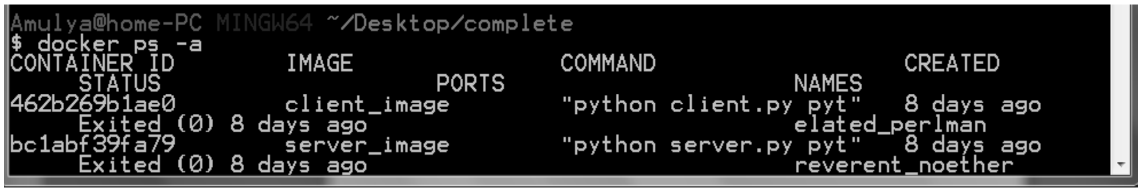
Fig 3.9.4   Images command example

➢ **ps**:List containers

Syntax:

docker ps[OPTIONS]

Example:



```
Amulya@home-PC MINGW64 ~/Desktop/complete
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
     STATUS                     PORTS               NAMES
462b269b1ae0        client_image        "python client.py pyt"   8 days ago
     Exited (0) 8 days ago                          elated_perlman
bc1abf39fa79        server_image        "python server.py pyt"   8 days ago
     Exited (0) 8 days ago                          reverent_noether
```

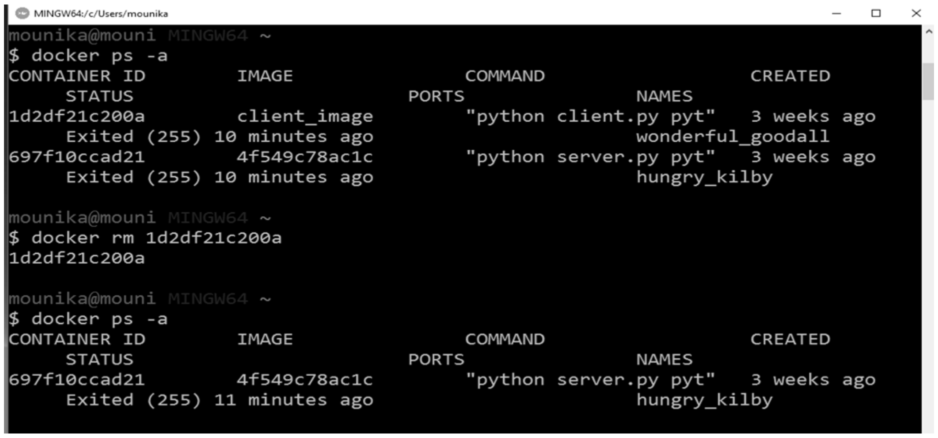
Fig 3.9.5   ps command example

➢ **rm**: Remove one or more containers.

Syntax:

docker rm [OPTIONS]   CONTAINER [CONTAINER...]

Example:



```
mounika@mouni MINGW64 ~
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
     STATUS                     PORTS               NAMES
1d2df21c200a        client_image        "python client.py pyt"   3 weeks ago
     Exited (255) 10 minutes ago                    wonderful_goodall
697f10ccad21        4f549c78ac1c        "python server.py pyt"   3 weeks ago
     Exited (255) 10 minutes ago                    hungry_kilby

mounika@mouni MINGW64 ~
$ docker rm 1d2df21c200a
1d2df21c200a

mounika@mouni MINGW64 ~
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
     STATUS                     PORTS               NAMES
697f10ccad21        4f549c78ac1c        "python server.py pyt"   3 weeks ago
     Exited (255) 11 minutes ago                    hungry_kilby
```

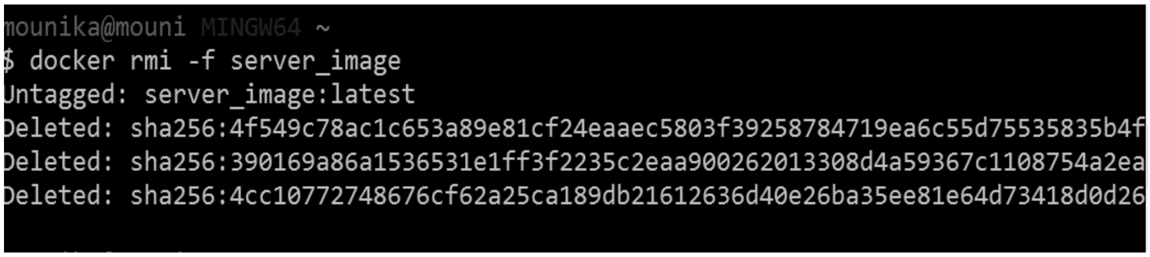
Fig 3.9.6 rm command example

- **rmi**: Remove one or more images

  Syntax:

  docker rmi [OPTIONS] IMAGE [IMAGE...]

  Example:



```
mounika@mouni MINGW64 ~
$ docker rmi -f server_image
Untagged: server_image:latest
Deleted: sha256:4f549c78ac1c653a89e81cf24eaaec5803f39258784719ea6c55d75535835b4f
Deleted: sha256:390169a86a1536531e1ff3f2235c2eaa900262013308d4a59367c1108754a2ea
Deleted: sha256:4cc10772748676cf62a25ca189db21612636d40e26ba35ee81e64d73418d0d26
```

Fig 3.9.7 rmi command example

Commands needed for creating a Docker file :

- **FROM**: The FROM instruction sets the Base Image for subsequent instructions.

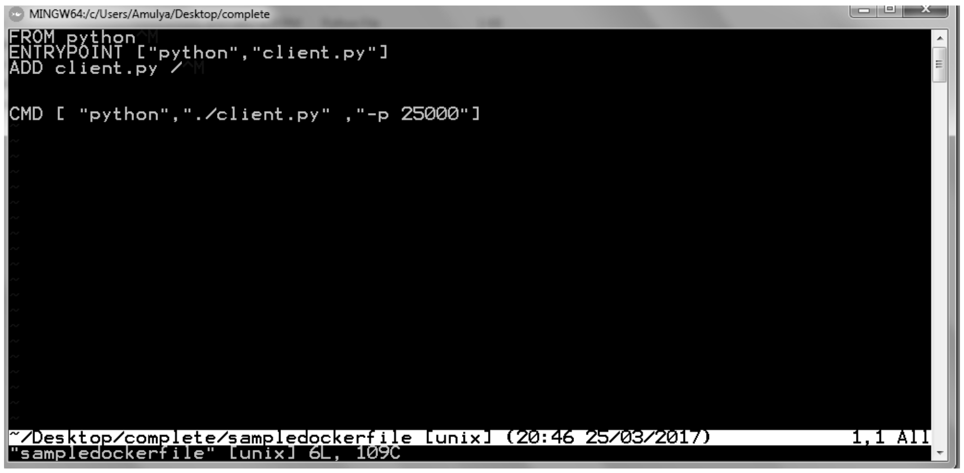
  Syntax: FROM <image>:<tag>
- **ADD:** The ADD instruction copies new files, directories remote file URLs from <src> and adds them to the filesystem of the image at the path <dest>.

  Syntax: ADD<src>... <dest>
- **ENTRY POINT**: An ENTRYPOINT allows you to configure a container that will run as an executable.

  Syntax: ENTRYPOINT ["executable","param1","param2"]
- **CMD**: The main purpose of a CMD is to provide defaults for an executing container .

  Syntax: CMD ["executable","param1","param2"]

  Example :



```
MINGW64:/c/Users/Amulya/Desktop/complete
FROM python
ENTRYPOINT ["python","client.py"]
ADD client.py /

CMD [ "python","./client.py" ,"-p 25000"]

~/Desktop/complete/sampledockerfile [unix] (20:46 25/03/2017)        1,1 All
"sampledockerfile" [unix] 6L, 109C
```

Fig 3.9.8 Sample Docker file.

CHAPTER-4
DEPLOYING PYTHON APPLICATION IN DOCKER CONTAINERS

4.1 MISSION OF DOCKER

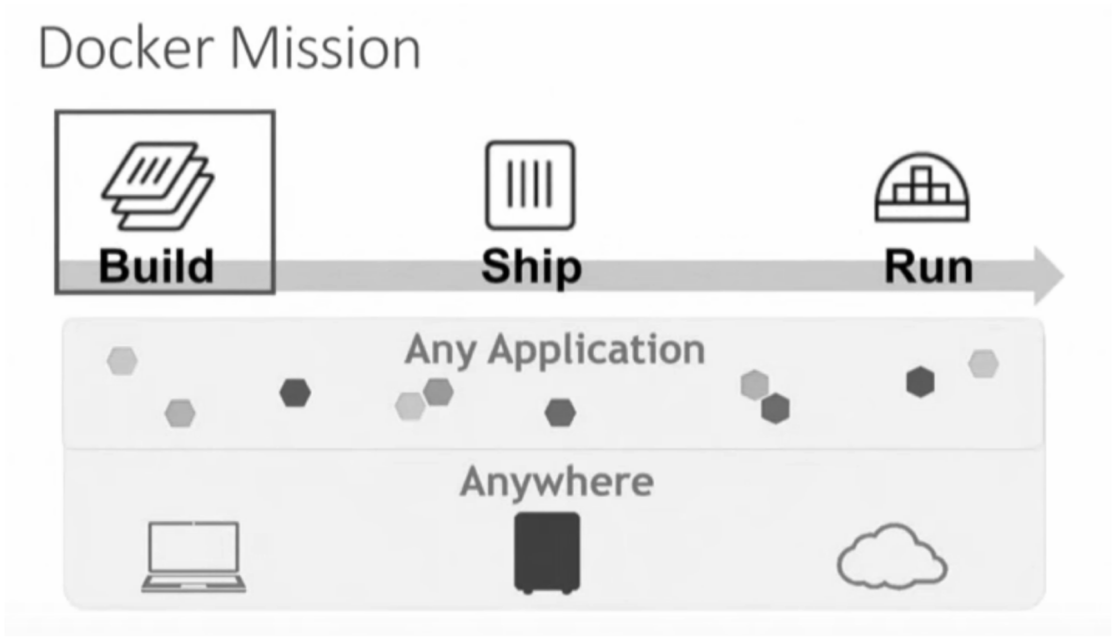


Fig 4.1 Docker Mission

Docker enables us to build our application in a certain platform, ship and run our application on any platform. Docker supports building of any kind of application and it can be run on any kind of platform once it is shipped.

4.2 STEPS TO CREATE A CONTAINER AND RUN IN DOCKER

There are certain steps to create a container and make our application run in docker. They are :

1.Write a Dockerfile: In this step, you use a text editor to write a short Dockerfile. A Dockerfile is a recipe which describes the files, environment, and commands that make up an image.

2.Build an image from your Dockerfile: While you are in the mydockerbuild directory, build the image using the docker build command.

3.Run your new docker-whale: Run your new image by typing docker run image-name.

4.3 CREATING A DOCKER FILE

In this step, you use a text editor to write a short Docker file. A Docker file is a recipe which describes the files, environment, and commands that make up an image. Your recipe is going to be very short.

You run these steps in a terminal window on Linux or macOS, or a command prompt on Windows. Remember that if you are using macOS or Windows, you are still creating an image which runs on Linux.

1. Make a new directory. If you're on Windows, use md instead of mkdir.

This directory will contain all the things you need to build your image. Right now, it's empty.

```
$ mkdir mydockerbuild
```

2. Change to your new directory. Whether you're on Linux, macOS, or Windows, the cd command is the same.

```
$ cd mydockerbuild
```

3. Edit a new text file named Dockerfile in the current directory, using a text editor such as nano or vi on Linux or Mac, or notepad on Windows.

```
$ nano Dockerfile
```

Linux or Mac:

Windows: The . tells notepad not to add a .txt extension.

```
C:\> notepad Dockerfile
```

4. Add a FROM statement by copying the following line into the file:

```
FROM docker/whalesay:latest
```

The FROM keyword tells Docker which image your image is based on. Whalesay is cute and has the cowsay program already, so we'll start there.

5. Add a RUN statement which will install the fortunes program into the image.

```
RUN apt-get -y update && apt-get install -y fortunes
```

The whalesay image is based on Ubuntu, which uses apt-get to install packages. These two commands refresh the list of packages available to the image and install the fortunes program into it. The fortunes program prints out wise sayings for our whale to say.

6. Add a CMD statement, which tells the image the final command to run after its environment is set up. This command runs fortune -a and sends its output to the cowsay command.

```
CMD /usr/games/fortune -a | cowsay
```

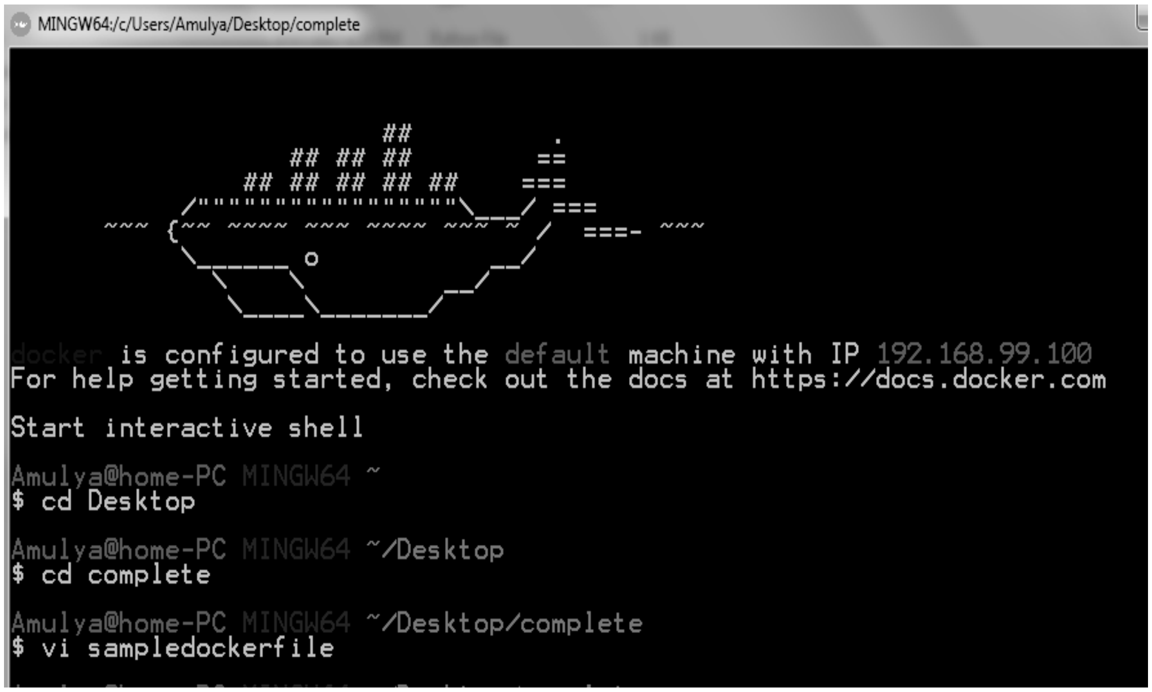
7.Check your work. Your file should look just like this:

```
FROM docker/whalesay:latest RUN apt-get -y update && apt-get install -y fortunes
CMD /usr/games/fortune -a | cowsay
```

8.Save the file and close the text editor. At this point, your software recipe is described in the Docker file file. You are ready to build a new image.

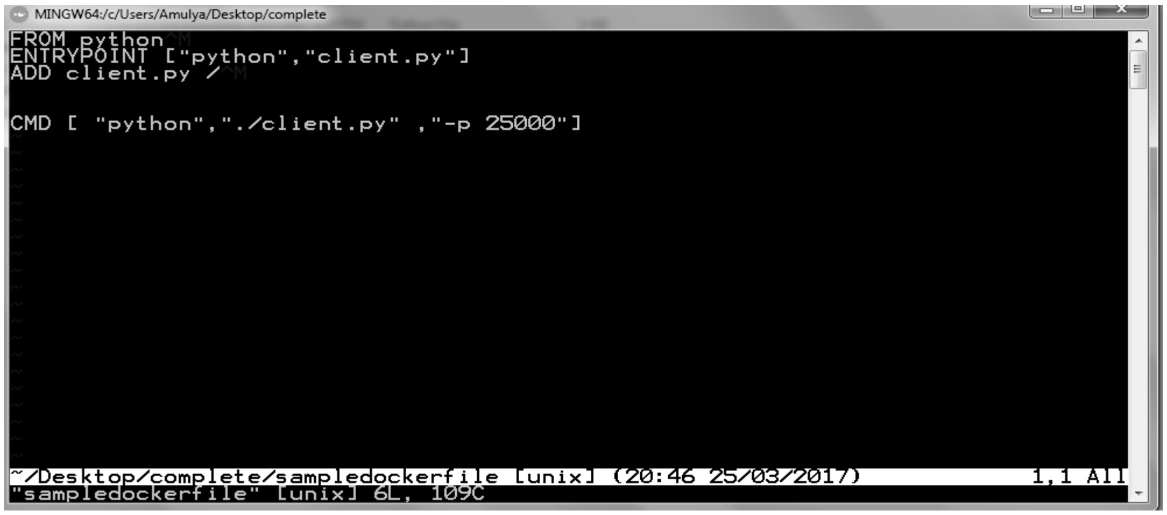
4.3.1 HOW TO OPEN A DOCKER FILE

We use vi command to open a docker file



```
                        ##         .
                  ## ## ##        ==
               ## ## ## ## ##    ===
           /"""""""""""""""""\___/ ===
      ~~~ {~~ ~~~~ ~~~ ~~~~ ~~~ ~ /  ===- ~~~
           \______ o           __/
             \    \         __/
              \____\_______/

docker is configured to use the default machine with IP 192.168.99.100
For help getting started, check out the docs at https://docs.docker.com

Start interactive shell

Amulya@home-PC MINGW64 ~
$ cd Desktop

Amulya@home-PC MINGW64 ~/Desktop
$ cd complete

Amulya@home-PC MINGW64 ~/Desktop/complete
$ vi sampledockerfile
```

Fig 4.3.1.1 Opening a docker file.



```
FROM python^M
ENTRYPOINT ["python","client.py"]
ADD client.py /^M

CMD [ "python","./client.py" ,"-p 25000"]
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~/Desktop/complete/sampledockerfile [unix] (20:46 25/03/2017)          1,1 All
"sampledockerfile" [unix] 6L, 109C
```

Fig 4.3.1.2 Example Docker file

➢ Now the docker file is opened .To close the docker file use :wq command by pressing double escape

4.4 BUILDING AN IMAGE

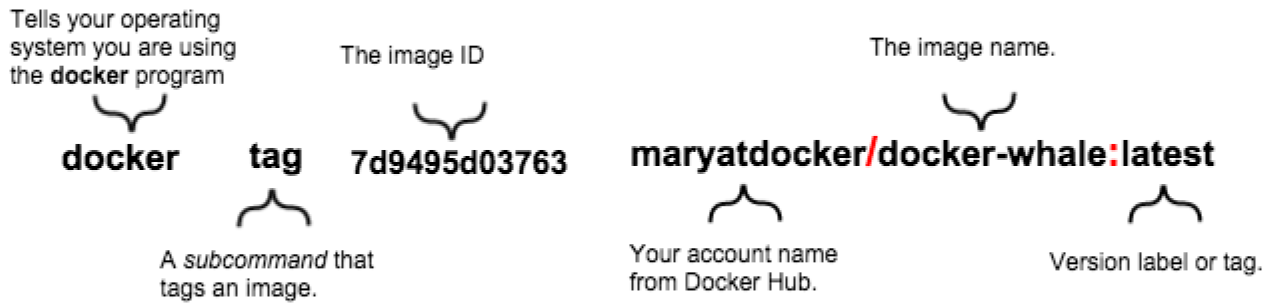
While you are in the mydockerbuild directory, build the image using the docker build command. The -t parameter gives your image a tag, so you can run it more easily later. Don't forget the . command, which tells the docker build command to look in the current directory for a file called Dockerfile. This command works the same in Linux, macOS, or Windows.

```
$ docker build -t docker-whale .

Sending build context to Docker daemon 2.048KB

...snip...

Removing intermediate container cb53c9d09f3b

Successfully built c2c3152907b5
```

The command takes several seconds to run and reports its outcome.

Tag the docker-whale image using the docker tag command and the image ID.

The command you type looks like this:

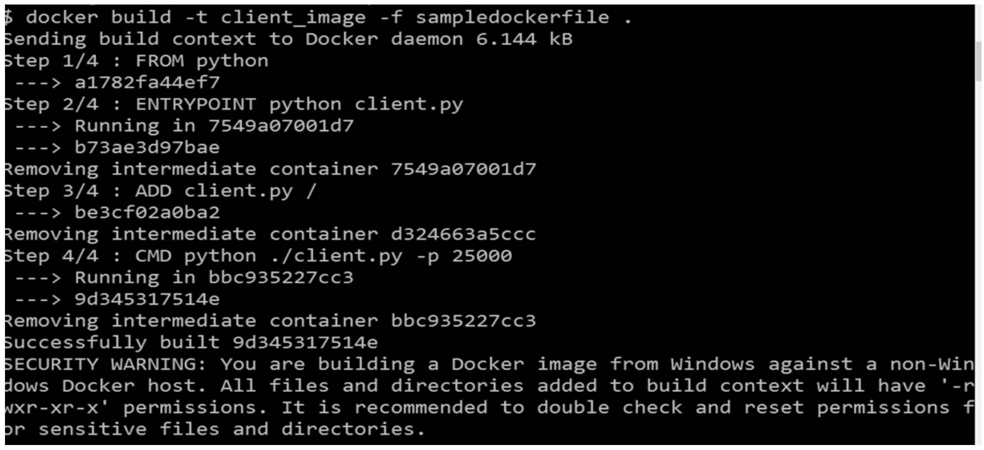


```
docker tag 7d9495d03763 maryatdocker/docker-whale:latest
```

Tells your operating system you are using the docker program
The image ID
The image name.
A subcommand that tags an image.
Your account name from Docker Hub.
Version label or tag.

Fig 4.4 Tag an image

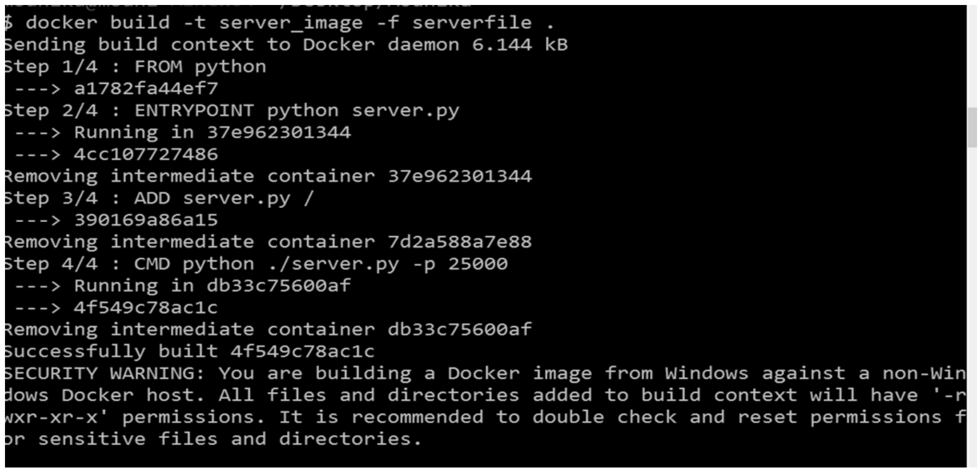
4.4.1 BUILDING CLIENT IMAGE



```
$ docker build -t client_image -f sampledockerfile .
Sending build context to Docker daemon 6.144 kB
Step 1/4 : FROM python
 ---> a1782fa44ef7
Step 2/4 : ENTRYPOINT python client.py
 ---> Running in 7549a07001d7
 ---> b73ae3d97bae
Removing intermediate container 7549a07001d7
Step 3/4 : ADD client.py /
 ---> be3cf02a0ba2
Removing intermediate container d324663a5ccc
Step 4/4 : CMD python ./client.py -p 25000
 ---> Running in bbc935227cc3
 ---> 9d345317514e
Removing intermediate container bbc935227cc3
Successfully built 9d345317514e
SECURITY WARNING: You are building a Docker image from Windows against a non-Win
dows Docker host. All files and directories added to build context will have '-r
wxr-xr-x' permissions. It is recommended to double check and reset permissions f
or sensitive files and directories.
```

Fig 4.4.1 Building client image

4.4.2 BUILDING SERVER IMAGE



```
$ docker build -t server_image -f serverfile .
Sending build context to Docker daemon 6.144 kB
Step 1/4 : FROM python
 ---> a1782fa44ef7
Step 2/4 : ENTRYPOINT python server.py
 ---> Running in 37e962301344
 ---> 4cc107727486
Removing intermediate container 37e962301344
Step 3/4 : ADD server.py /
 ---> 390169a86a15
Removing intermediate container 7d2a588a7e88
Step 4/4 : CMD python ./server.py -p 25000
 ---> Running in db33c75600af
 ---> 4f549c78ac1c
Removing intermediate container db33c75600af
Successfully built 4f549c78ac1c
SECURITY WARNING: You are building a Docker image from Windows against a non-Win
dows Docker host. All files and directories added to build context will have '-r
wxr-xr-x' permissions. It is recommended to double check and reset permissions f
or sensitive files and directories.
```

Fig 4.4.2 Building server image

4.5 HOW TO RUN A CONTAINER

Now you can verify that the new image is on your computer and you can run it.In a terminal window or command prompt, type docker images, which lists the images you have locally.

```
$ docker images

REPOSITORY          TAG        IMAGE ID          CREATED            SIZE

docker-whale         latest      c2c3152907b5     4 minutes ago      275.1 MB

docker/whalesay      latest      fb434121fc77     4 hours ago        247 MB

hello-world          latest      91c95931e552     5 weeks ago        910 B
```

Run your new image by typing docker run docker-whale

```
$ docker run [options :] docker-whale ___________________________________

< You will be successful inyour work. >-------------------------------------

##        .

## ## ##       ==

## ## ## ##      ===

      /""""""""""""""""""""""""""""___/ ===

  ~~~ {~~ ~~~~ ~~~ ~~~~ ~~ ~ /  ===- ~~~

      \______ o          __/

       \    \        __/

 \____\______/
```

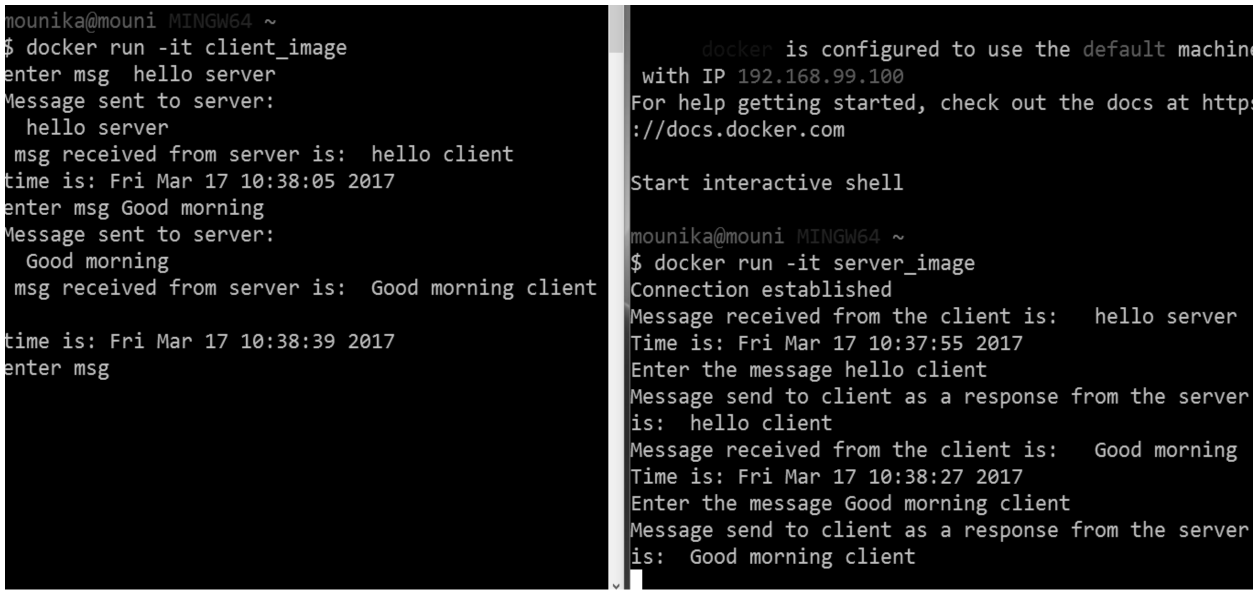
OPTIONS :

➢ **-it :**

In docker we have terminal type specified by tty. It is used to take dynamic input from command prompt in docker. By default the tty mode is in off state.To take input from command prompt we have to make tty mode enabled.

The whale is now a lot smarter. It comes up with its own thoughts and it takes less typing to run. You may also notice that Docker didn't have to download anything, because you built the image locally.

4.5.1 RUNNING A CONTAINER



```
mounika@mouni MINGW64 ~
$ docker run -it client_image
enter msg  hello server
Message sent to server:
  hello server
 msg received from server is:  hello client
time is: Fri Mar 17 10:38:05 2017
enter msg Good morning
Message sent to server:
  Good morning
 msg received from server is:  Good morning client

time is: Fri Mar 17 10:38:39 2017
enter msg
```

```
      docker is configured to use the default machin
 with IP 192.168.99.100
For help getting started, check out the docs at http
://docs.docker.com

Start interactive shell

mounika@mouni MINGW64 ~
$ docker run -it server_image
Connection established
Message received from the client is:   hello server
Time is: Fri Mar 17 10:37:55 2017
Enter the message hello client
Message send to client as a response from the server
is:  hello client
Message received from the client is:   Good morning
Time is: Fri Mar 17 10:38:27 2017
Enter the message Good morning client
Message send to client as a response from the server
is:  Good morning client
```

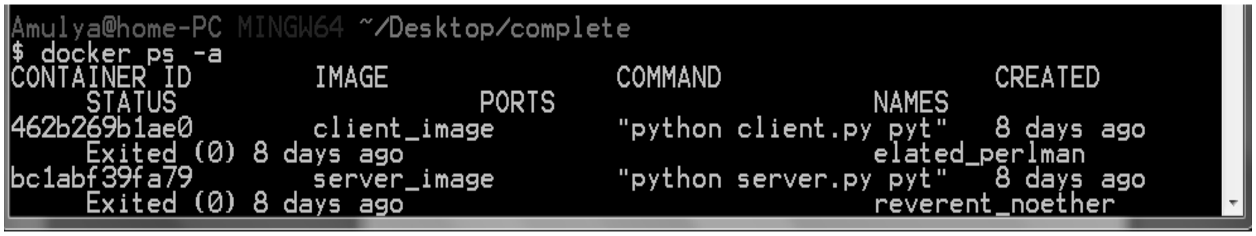
Fig 4.5.1 Client Server implementation in Docker Containers

4.6 DOCKER LOGS

Syntax:

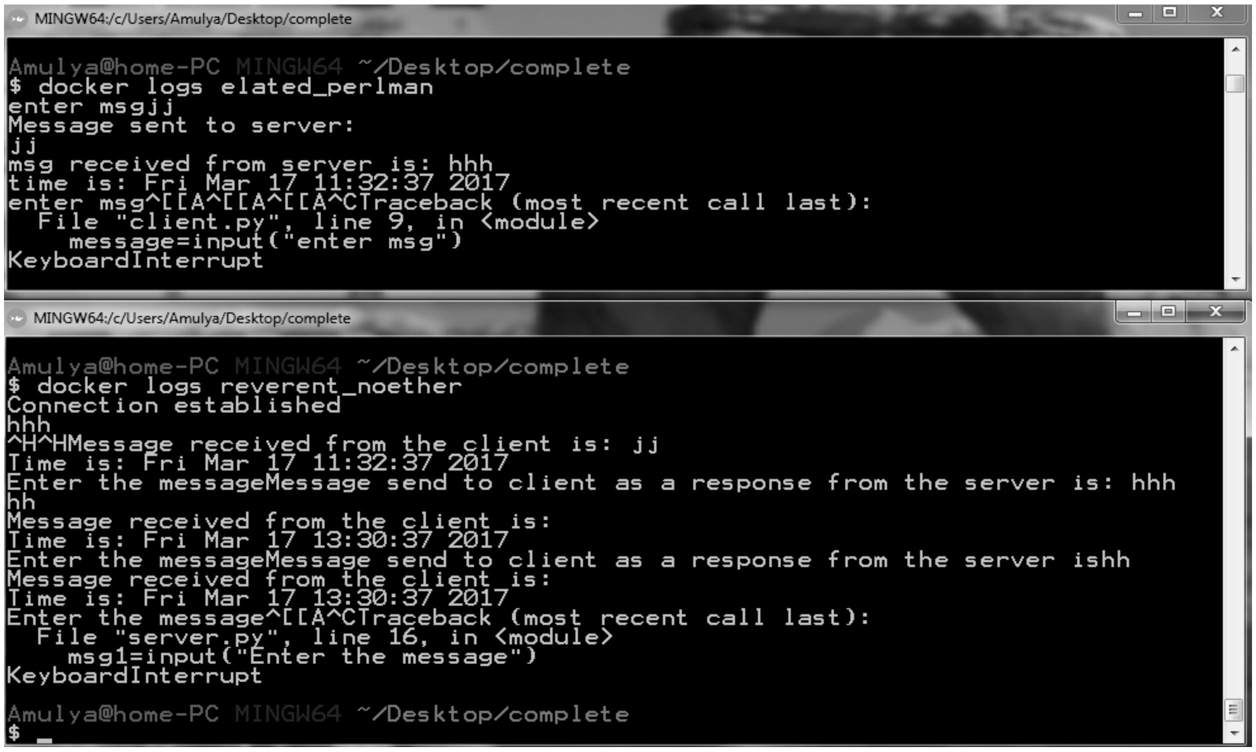
docker logs container_name

NOTE: Container name can be known using 'docker ps –a' command



```
Amulya@home-PC MINGW64 ~/Desktop/complete
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS                     PORTS               NAMES
462b269b1ae0        client_image        "python client.py pyt"   8 days ago
    Exited (0) 8 days ago                          elated_perlman
bc1abf39fa79        server_image        "python server.py pyt"   8 days ago
    Exited (0) 8 days ago                          reverent_noether
```

Fig 4.6.1   Listing containers



```
MINGW64:/c/Users/Amulya/Desktop/complete

Amulya@home-PC MINGW64 ~/Desktop/complete
$ docker logs elated_perlman
enter msgjj
Message sent to server:
jj
msg received from server is: hhh
time is: Fri Mar 17 11:32:37 2017
enter msg^[[A^[[A^[[A^CTraceback (most recent call last):
  File "client.py", line 9, in <module>
    message=input("enter msg")
KeyboardInterrupt
```

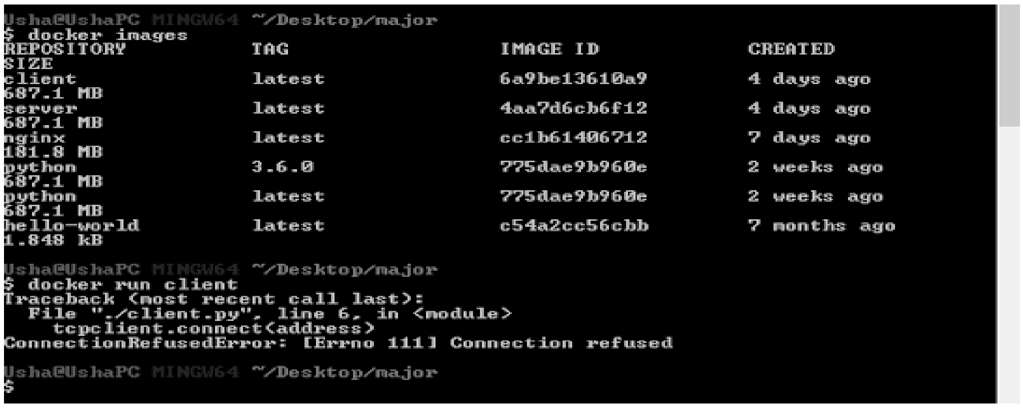
```
MINGW64:/c/Users/Amulya/Desktop/complete

Amulya@home-PC MINGW64 ~/Desktop/complete
$ docker logs reverent_noether
Connection established
hhh
^H^HMessage received from the client is: jj
Time is: Fri Mar 17 11:32:37 2017
Enter the messageMessage send to client as a response from the server is: hhh
hh
Message received from the client is:
Time is: Fri Mar 17 13:30:37 2017
Enter the messageMessage send to client as a response from the server ishh
Message received from the client is:
Time is: Fri Mar 17 13:30:37 2017
Enter the message^[[A^CTraceback (most recent call last):
  File "server.py", line 16, in <module>
    msg1=input("Enter the message")
KeyboardInterrupt

Amulya@home-PC MINGW64 ~/Desktop/complete
$ _
```

Fig 4.6.2   Output

4.7 TEST CASES

Test case 1:

Here in Fig 4.7.1, we did not mention the server IP address to client. So ,we got connection refused error. We have to mention the IP address of server to client in python program so that connection is established between them.

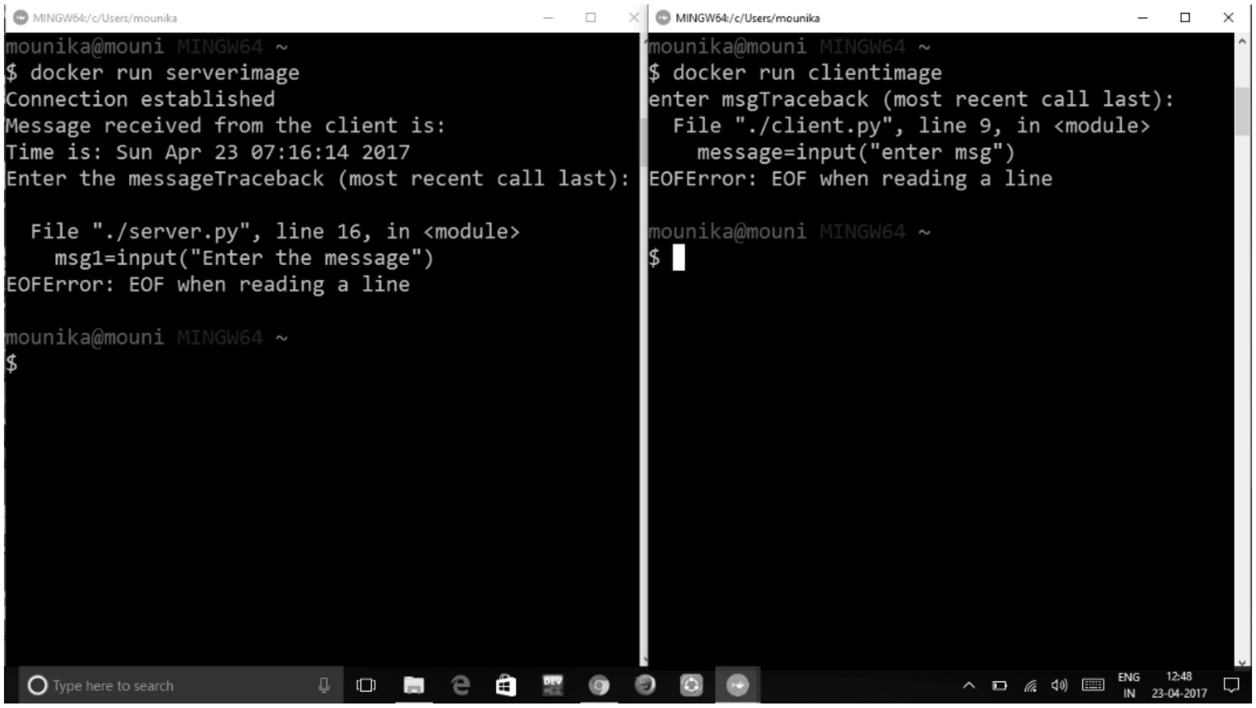


```
Usha@UshaPC MINGW64 ~/Desktop/major
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
client              latest              6a9be13610a9        4 days ago
687.1 MB
server              latest              4aa7d6cb6f12        4 days ago
687.1 MB
nginx               latest              cc1b61406712        7 days ago
181.8 MB
python              3.6.0               775dae9b960e        2 weeks ago
687.1 MB
python              latest              775dae9b960e        2 weeks ago
687.1 MB
hello-world         latest              c54a2cc56cbb        7 months ago
1.848 kB

Usha@UshaPC MINGW64 ~/Desktop/major
$ docker run client
Traceback (most recent call last):
  File "./client.py", line 6, in <module>
    tcpclient.connect(address)
ConnectionRefusedError: [Errno 111] Connection refused

Usha@UshaPC MINGW64 ~/Desktop/major
$
```

Fig 4.7.1 Connection refused error

Test case 2:

In docker console by default the interactive mode will be in off mode. To activate interactive mode we should use –it option. Without using –it option dynamic input cannot be accepted .



```
mounika@mouni MINGW64 ~
$ docker run serverimage
Connection established
Message received from the client is:
Time is: Sun Apr 23 07:16:14 2017
Enter the messageTraceback (most recent call last):

  File "./server.py", line 16, in <module>
    msg1=input("Enter the message")
EOFError: EOF when reading a line

mounika@mouni MINGW64 ~
$
```

```
mounika@mouni MINGW64 ~
$ docker run clientimage
enter msgTraceback (most recent call last):
  File "./client.py", line 9, in <module>
    message=input("enter msg")
EOFError: EOF when reading a line

mounika@mouni MINGW64 ~
$
```

Fig 4.7.2 EOF error

CHAPTER-5

CONCLUSION AND FUTURE SCOPE

5.1 CONCLUSION

A working model of client server model using python in docker container has been successfully implemented. Client and server are able to exchange between them and the time of exchanging is also produced. The same is deployed in Docker container and the output is saved using the docker logs command so that the output cannot be changed by any external source and is saved in the docker.

The biggest advantage of our project is that the client server model can be implemented in docker which can be deployed on any platform which saves maximum amount of execution time of our applications. Applications are isolated and independent of the platform being deployed on.

5.2 FUTURE SCOPE

We have implemented client server implementation in docker containers using command prompt of docker. We can implement this application in graphical user interface. We can implement the client server implementation model with more than one client also. In client server application, input is taken by the interactive mode of docker. We can take the input from the forms also.

APPENDIX

➢ DESCRIPTION

Client and server application should be developed in python and establish communication between them with sample messages along with time. Server and client application should run inside a docker container. Seamlessly they should be able to communicate with each other. Whenever there is communication the data transferred should be printed.

➢ SAMPLE CODES

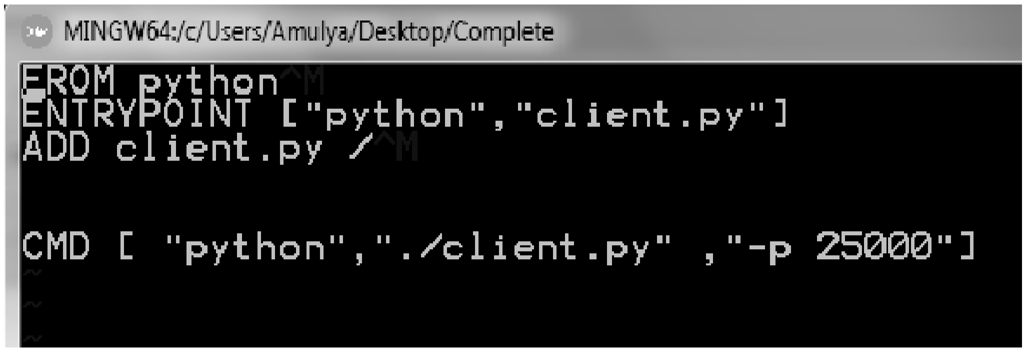
Sample server.py file

```
#!/usr/bin/python   # This is server.py file

import socket   # Import socket module

s = socket.socket() # Create a socket object

host = socket.gethostname() # Get local machine name

port =12345 # Reserve a port for your service.

s.bind((host, port)) # Bind to the port

s.listen(5) # Now wait for client connection.whileTrue:

c, addr = s.accept() # Establish connection with client.

print'Got connection from', addr

 c.send('Thank you for connecting')

 c.close() # Close the connection
```

Sample Client.py file

```
#!/usr/bin/python   # This is client.py file

import socket   # Import socket module

s = socket.socket() # Create a socket object

host = socket.gethostname() # Get local machine name

port =12345 # Reserve a port for your service.

s.connect((host, port))

print s.recv(1024)

s.close  # Close the socket when done
```

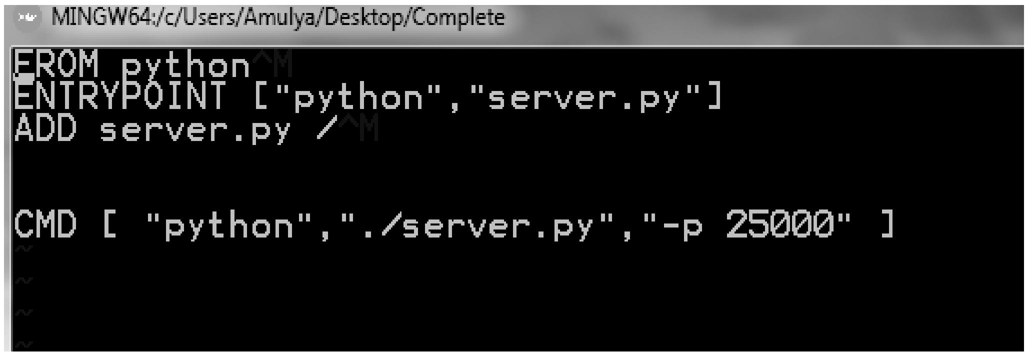
Server Docker File



```
MINGW64:/c/Users/Amulya/Desktop/Complete
FROM python^M
ENTRYPOINT ["python","client.py"]
ADD client.py /^M


CMD [ "python","./client.py"  ,"-p 25000"]
~
~
~
```

Fig 1 Server Docker file

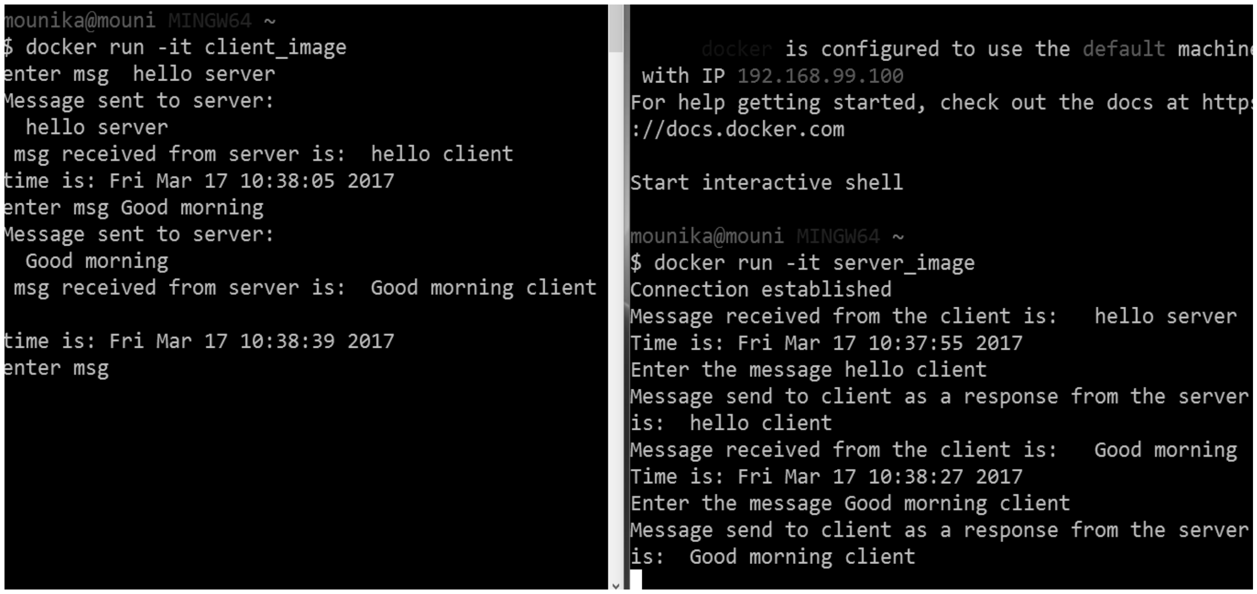
Client Docker File



```
MINGW64:/c/Users/Amulya/Desktop/Complete
FROM python^M
ENTRYPOINT ["python","server.py"]
ADD server.py /^M


CMD [ "python","./server.py","-p 25000" ]
~
~
~
~
```

Fig 2 Client docker file

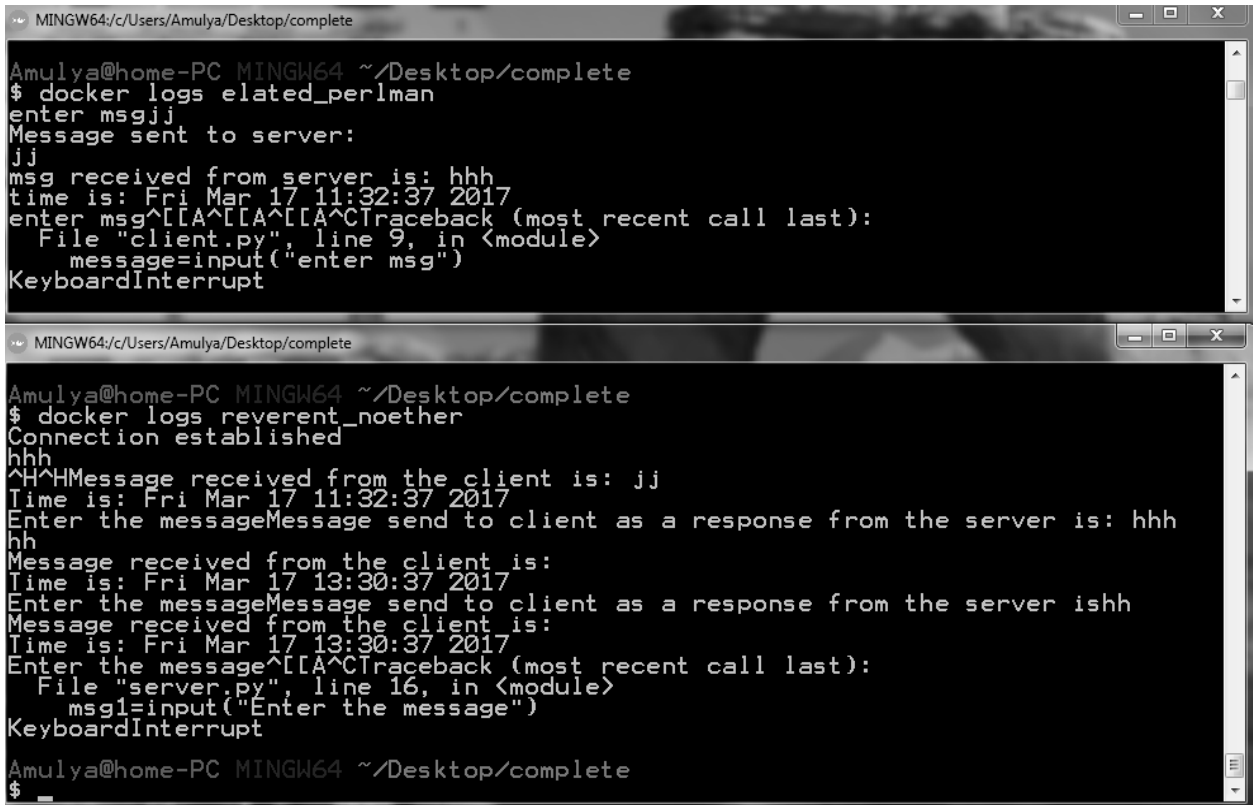
➢ OUTPUT



```
mounika@mouni MINGW64 ~
$ docker run -it client_image
enter msg  hello server
Message sent to server:
  hello server
 msg received from server is:  hello client
time is: Fri Mar 17 10:38:05 2017
enter msg Good morning
Message sent to server:
  Good morning
 msg received from server is:  Good morning client

time is: Fri Mar 17 10:38:39 2017
enter msg
```

```
      docker is configured to use the default machin
 with IP 192.168.99.100
For help getting started, check out the docs at http
://docs.docker.com

Start interactive shell

mounika@mouni MINGW64 ~
$ docker run -it server_image
Connection established
Message received from the client is:   hello server
Time is: Fri Mar 17 10:37:55 2017
Enter the message hello client
Message send to client as a response from the server
is:  hello client
Message received from the client is:   Good morning
Time is: Fri Mar 17 10:38:27 2017
Enter the message Good morning client
Message send to client as a response from the server
is:  Good morning client
```

Fig 3 Client server communication in docker



MINGW64:/c/Users/Amulya/Desktop/complete

```
Amulya@home-PC MINGW64 ~/Desktop/complete
$ docker logs elated_perlman
enter msgjj
Message sent to server:
jj
msg received from server is: hhh
time is: Fri Mar 17 11:32:37 2017
enter msg^[[A^[[A^[[A^CTraceback (most recent call last):
  File "client.py", line 9, in <module>
    message=input("enter msg")
KeyboardInterrupt
```

MINGW64:/c/Users/Amulya/Desktop/complete

```
Amulya@home-PC MINGW64 ~/Desktop/complete
$ docker logs reverent_noether
Connection established
hhh
^H^HMessage received from the client is: jj
Time is: Fri Mar 17 11:32:37 2017
Enter the messageMessage send to client as a response from the server is: hhh
hh
Message received from the client is:
Time is: Fri Mar 17 13:30:37 2017
Enter the messageMessage send to client as a response from the server ishh
Message received from the client is:
Time is: Fri Mar 17 13:30:37 2017
Enter the message^[[A^CTraceback (most recent call last):
  File "server.py", line 16, in <module>
    msg1=input("Enter the message")
KeyboardInterrupt

Amulya@home-PC MINGW64 ~/Desktop/complete
$ _
```

Fig 4 Docker logs Output

REFERENCES

1. www.docker.com
2. Byron Francis, “Docker: The Complete Beginner's Guide” , 1st Edition, Createspace independent publishing platform.
3. Darryl Barton, “Docker: A Comprehensive Beginner's Guide”, 1st Edition, Createspace independent publishing platform.
4. Adrian Mouat, “Using Docker: Developing and Deploying Software with Containers”, 3rd Edition, O’Reilly Media.
5. www.docs.docker.com
6. www.digitalocean.com
7. David Ascher and Mark Lutz, "Learning python" , 5th Edition, O’Reilly Media.
8. Magnus Lie Hetland, "Beginning python”, 2nd Edition, Apress.
9. John Goerzen, "Foundations Of Python Network Programming",2nd Edition, Apress.
10. Ross Dawson, "Python Programming for the Absolute Beginner", 3rd Edition, Cengage.
11. https://www.tutorialspoint.com/python/python_networking.htm
12. http://www.bogotobogo.com/python/python_network_programming_server_client.php
13. https://www.python.org/about/gettingstarted/
14. https://www.pluralsight.com/blog/it-ops/networking-basics-tcp-udp-tcpip-osi-models
15. Andrew S. Tanenbaum, " Computer Networks", 5th Edition, Pearson.