**School of Information Sciences**

Term Project Report

**DOCKER MANAGEMENT USING ANSIBLE**

|  |  |  |  |
| --- | --- | --- | --- |
| **Registration Number** | **Name** | **Branch** | **E-mail** |
| 1XXXXXXX4 | Anand D Rao | Computing Technology & Virtualization | xxxxxxxx@yahoo.com |
| 1XXXXXXX6 | Sohaib Mohammed Rafique | xxxxxxxxxxxx@gmail.com |

**Guide/Panel Team**

|  |
| --- |
| **Name** |
| Mr. Mohan Kumar J |
| Mr. Satyanarayan Shenoy |
| Mr. Samarendranath Bhattacharya |

**Version 1.0**

**Submitted on**

**6th November 2017**

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| 1 | Introduction | 7 |
| 1.1 | Literature Review | 8 |
| 2 | Workplan | 10 |
| 3 | Requirement Specification | 11 |
| 3.1 | Software Requirements | 11 |
| 3.2 | Hardware Requirements | 11 |
| 3.3 | Quality Requirements | 11 |
| 4 | Design | 12 |
| 4.1 | Block Diagram | 12 |
| 4.2 | Unit Wise Description | 13 |
| 5 | Implementation | 14 |
| 5.1 | Package Used Description | 14 |
| 5.2 | Steps carried out in project implementation | 14 |
| 6 | Test Documentation | 18 |
| 6.1 | MySQL Server Test | 18 |
| 6.2 | Nginx Server Test | 19 |
| 6.3 | Proxy Test | 20 |
| 7 | References | 21 |
| 7.1 | Bibliography | 21 |
| 7.2 | Webliography | 21 |
|  | Appendix | 22 |
| A | Source Code | 22 |
|  |  |  |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| Figure 1.1 | Docker Architecture | 8 |
| Figure 4.1 | Interaction between the Master Computer and Host Computer | 12 |
| Figure 4.2 | Docker Functional Component | 12 |
| Figure 6.1 | Successful Execution of Ansible Playbook sql.yml | 18 |
| Figure 6.2 | Docker containers running on the host machine | 18 |
| Figure 6.3 | Docker Container running MySQL server | 19 |
| Figure 6.4 | Successful Execution of Ansible Playbook nginx.yml | 19 |
| Figure 6.5 | Docker containers running on the host machine | 20 |
| Figure 6.6 | Webpage accessed through the nginx server | 20 |
| Figure 6.7 | Proxy Failed Test | 20 |

**ABSTRACT**

Docker is the most popular file format for Linux-based container development and deployments. It 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.

When used in brand-new environments, Docker containers can be managed easily. But it can be a challenge to manage Docker containers along with tools which are required to manage IT environments. With the help of a powerful tool like Ansible, this problem can be easily solved. Ansible is an IT automation engine that handles tasks such as automating configuration management, application deployment, intra-service orchestration.

Ansible helps in automation of building docker containers and in the deployment process. It can not only manage the containers, but also the environments around the containers. Docker containers run on hosts which need to be launched, configured, networked, and coordinated, whether they are local machines or full cloud infrastructures.

**MY CONTRIBUTION**

Deployment of docker containers in host machine requires a lot of pre-requisites, setting up the required environment for the smooth functioning of it. My contribution towards the project was to set up this environment with pre-requisites.

1)Searching for an application which can be better for showcasing the docker features and pre-requisites required for docker container set up.

2)Coming up with a Dockerfile which has all the pre-requisites for deploying the application on the host machine. Defining Dockerfiles for installing MySQL and Nginx server in the host containers.

3)Define the proxy settings required for smooth running of application and docker containers in the host machine. And making certain changes to config files of docker as well as containers config files.

4)Testing under different conditions to ensure application runs smoothly.

**PROJECT OBJECTIVE**

The objective of the project is to deploy Docker Container having Ubuntu 16.04 as the base image to host computers with the help of Ansible. Also, Ansible will be used to install Nginx and MySQL servers on the docker containers.

**CHAPTER – 1**

**INTRODUCTION**

Not very often does a technology come along that is so widely adopted across an entire industry in short duration of time. Docker is an open-source engine that automates the deployment of applications into containers. It enables separation of applications from the infrastructure, so that the software can be delivered quickly.

Since the public release of Docker in March 2013, it has managed to solve major problems commonly faced in the development. The first issue is aimed at the fact that Virtual Machines are large-weight compute resources. A virtual machine would normally take couple of seconds to launch, while docker containers can launch in sub seconds. Removing the overhead of the hypervisor also means containers are highly efficient and multiple containers can run simultaneously on the hosts and make the best use of the resources.

Another major issue faced was that when developers build applications and ship them to other departments, they discovered that the applications don’t run in same manner as it did on their workstations. With help of docker, developers can now build an application once, and it will run across every environment along the application lifecycle — from development to QA and eventually into production. Docker containers make it easier to package your application into a single image that’s easy to deploy in different places, which is why the Docker project has embraced the metaphor of the shipping container. Many organizations are using Docker as an integral part of their DevOps workflows because of their flexibility and portability.

When used in brand-new environments, Docker containers can be managed easily. But it can be a challenge to manage Docker containers along with tools which are required to manage IT environments. With the help of a powerful tool like Ansible, this problem can be easily solved. Ansible is an IT automation engine that handles tasks such as automating configuration management, application deployment, intra-service orchestration. It is designed to be minimal in nature, consistent, secure, and highly reliable. It has an extremely low learning curve such that such that new users can be quickly brought into new IT projects, and longstanding automation content can easily be understood even after months of being away from a project.

Ansible can configure needed services and deploy multi-tier applications reliably and consistently from one common framework. Ansible uses 'Playbooks' to achieve the desired state for a system. Playbooks are simple to write and maintain and make installations, upgrades and day-to-day management repeatable and reliable.

Ansible helps in automation of building docker containers and in the deployment process. It can not only manage the containers, but also the environments around the containers. Docker containers run on hosts which need to be launched, configured, networked, and coordinated, whether they are local machines or full cloud infrastructures. Management of both containerized and non-containerized applications is especially important as containerized application are always communicating with non-containerized applications such as storage, database, networking, etc.

* 1. **LITERATURE REVIEW**

Containers have a long and storied history in computing. Unlike hypervisor virtualization, where one or more independent machines run virtually on physical hardware via an intermediation layer, containers instead run in user space on top of an operating system’s kernel. As a result, container virtualization is often called operating system-level virtualization. Container technology allows multiple isolated user space instances to be run on a single host.

Dockers are an open platform for developing, shipping along with other dependencies and running application. Dockers provide the ability to package and run an application in a loosely isolated environment called container. The features like isolation and security allow you to run many applications simultaneously. These Dockers can be deployed inside virtual machines also.

Dockers fall under OS-level virtualization in which host operating system will be at the base and the virtualization layers are on top of it and run as an application within the operating system. This virtualization layer offers a file system and kernel service abstraction layer which isolates resources from these containers and it ensures that these containers appear as a standalone server.

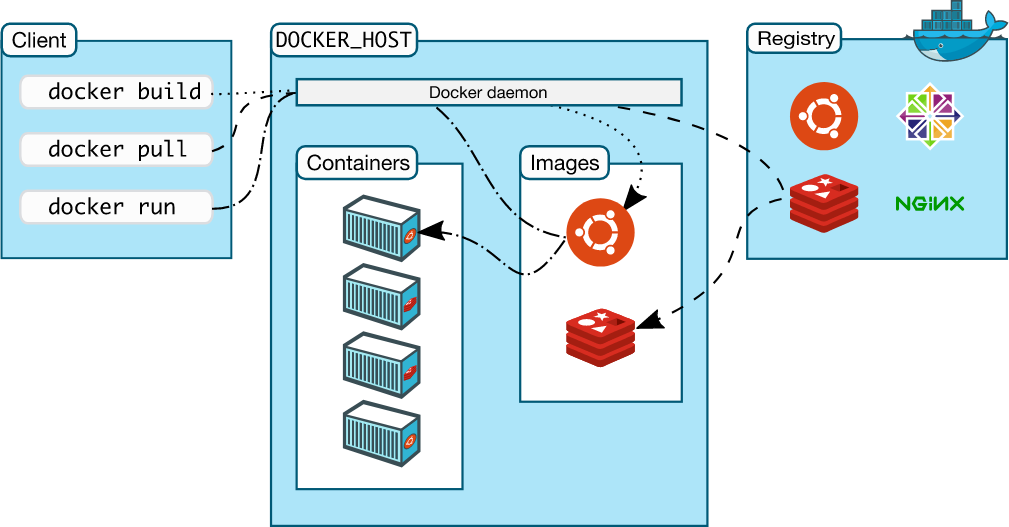


Figure 1.1: Docker Architecture

The docker daemon is responsible for creating and managing docker objects such as images, containers, networks and volumes. It listens to docker API and performs specified operations. These daemons communicate with each other to manage docker services. The docker client is the primary way that docker users interact with docker. When we execute commands such as docker run, the client sends these commands to daemon. The docker command uses docker API. These clients can communicate with more than one daemon.

The docker registers store docker images. There are many public registers like docker hub and docker cloud, and docker is configured to check for images on docker hub by default. When we use docker pull or docker run command, the required images are pulled from configured registry. When we use docker push command, the image is pushed to configured registry.

Docker image is a read-only template with instructions for creating a docker container. We can create our own image or use images that are created by others and published in a registry. To build our own image we need create dockerfile with a simple syntax for defining steps the steps needed to create an image and run it. Each instruction in dockerfile creates a layer in the container, when we change the dockerfile and rebuild it then only those layers which have been changed are rebuilt.

Docker containers is a runnable instance of an image. We can create, run, stop, move or delete a container using the docker API or CLI. We can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state. By default, containers are well isolated from other containers and from its host machine.

Ansible is an open source automation platform which can help with configuration management, application deployment, task automation. Ansible works by connecting to host machine and pushing out small programs, called "Ansible modules" to them. These programs are written to be resource models of the desired state of the system. Ansible then executes these modules and removes them when finished. Ansible uses YAML files as its main source of information at run time. YAML is a data representation language that is commonly used for configuration.

Ansible connects to the host machine by running commands via SSH, so there’s no need to pre-install an agent or any other software on the host. Ansible uses an “agentless” model where changes are pushed out to machines on demand. This is huge plus because the host machine can run the application without any CPU or memory-hungry daemons running in the background.

**CHAPTER – 2**

**WORK PLAN**

|  |  |
| --- | --- |
| WEEK | WORKPLAN |
| WEEK 1 | Literature Review |
| WEEK 2 | Understanding Docker and its dependencies |
| WEEK 3 | Understanding Ansible and its dependencies |
| WEEK 4 | Setting up Docker and Ansible & getting hands-on experience |
| WEEK 5 | Building Docker Images using Ansible for Nginx and MySQL servers |
| WEEK 6 | Running Docker Containers using Ansible for Nginx and MySQL servers |
| WEEK 7 | Deploying Docker Containers to Local Host |
| WEEK 8 | Solving Proxy Issue and Deploying Containers to remote hosts. |

**CHAPTER – 3**

**REQUIREMENTS SPECIFICATION**

**3.1) Software Requirements**

* Docker CE 17.03.0
* Ansible 2.4.0
* Ubuntu 16.04 LTS
* MySQL 14.14
* nginx 1.10.3
* Python 2
* docker - py
* docker - compose

**3.2) Hardware Requirements**

* Master Computer (For Deployment)
* Host Computers (To run the deployed Docker Containers)

**3.3) Quality Requirements**

**3.3.1) Master Computer**

* Install Docker CE 17.03 and Ansible 2.4.0.
* Install docker – py and docker – compose using pip.
* The Ansible host file should have information regarding the host machine - IP Address, username and password.
* Enable SSH and test connecting to the host computers.

**3.3.2) Host Computer**

* Create a sudo user called “test” for the main computer to connect through Ansible.
* Install Docker CE 17.03
* Install docker – py and docker – compose using pip.
* Create a docker group and add “test” user to it.
* Enable SSH

**CHAPTER 4**

**DESIGN**

**4.1) Block Diagram**

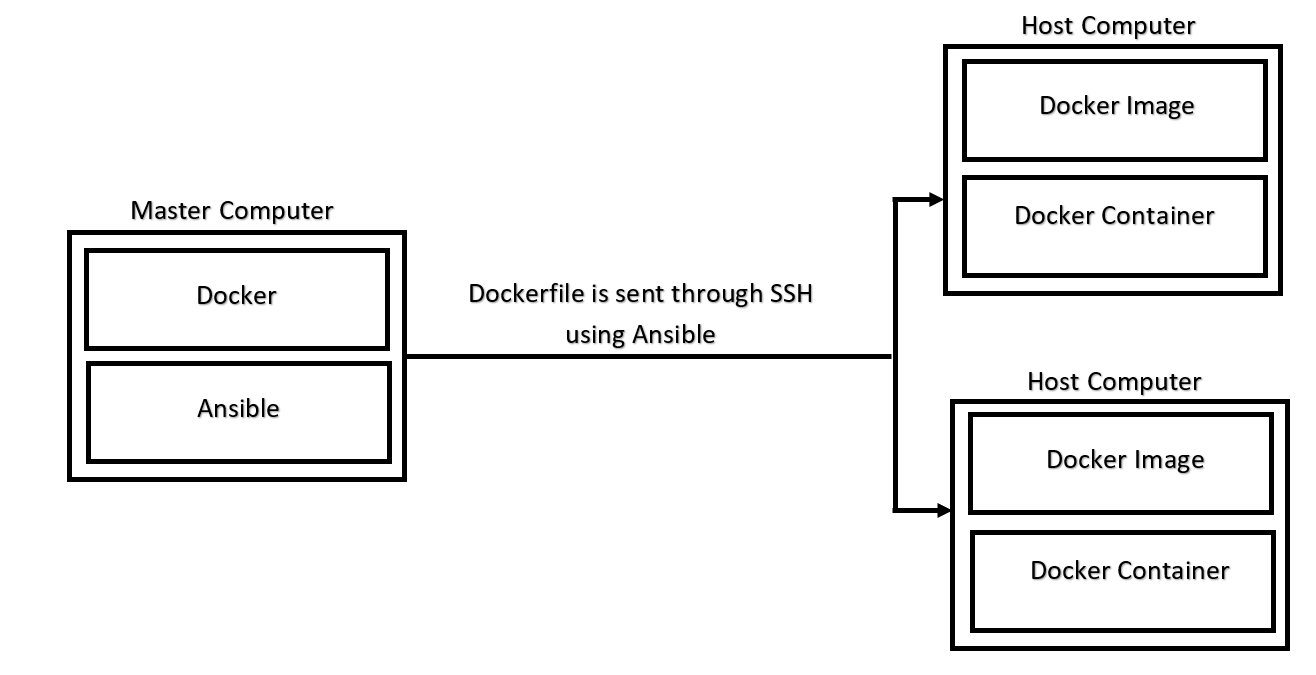
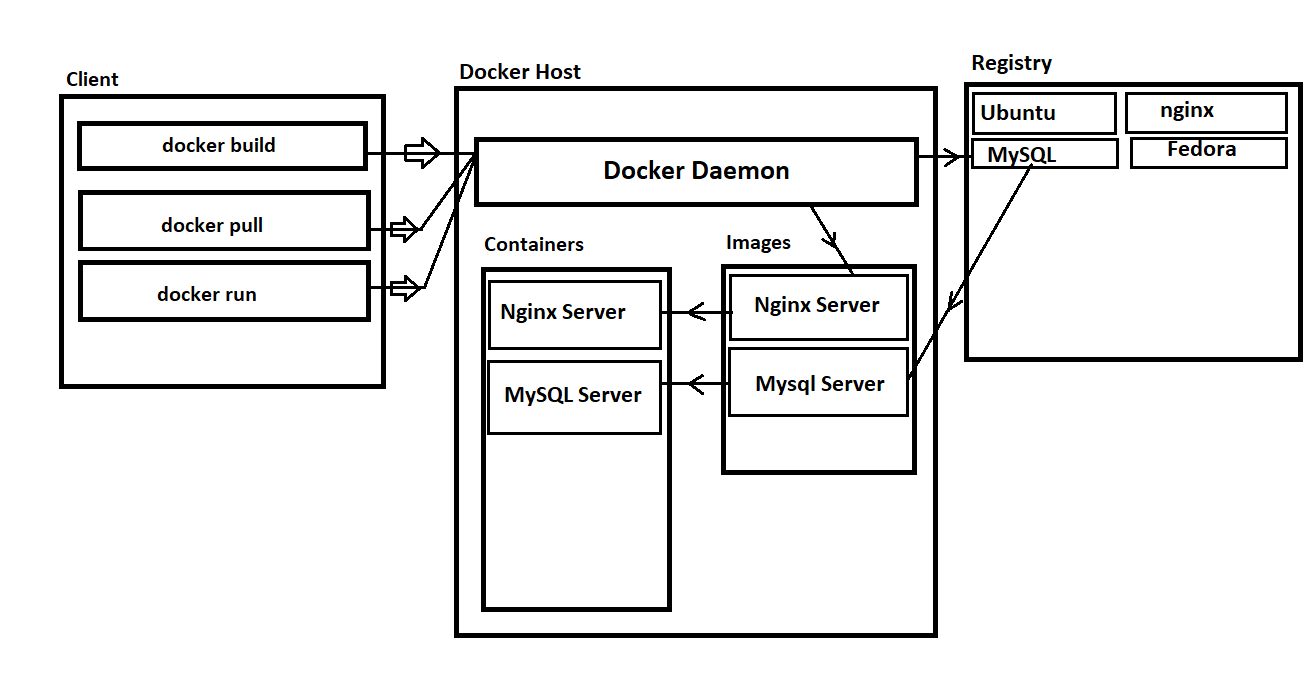
****

Figure 4.1: Interaction between the Master Computer and Host Computer

****  
Figure 4.2: Docker Functional Component

**4.2) Unit wise description**

**4.2.1) Master Computer**

The master computer is responsible for deploying docker containers using Ansible. It will have Docker and Ansible installed on it. Upon creating and running the Ansible playbook, it will connect to the host computer through SSH. The admin must add the remote host information in Ansible inventory file. Once the connection has been established, Ansible will copy the custom dockerfile to the host computer from the master computer. Ansible will then build the docker image from the dockerfile and run the docker container on the host computer.

**4.2.2) Host Computer**

The host computer will need to have SSH enabled and docker installed on it. Once the docker container has been created, the user can run the container on the host computer.

**4.2.3) Dockerfile**

To build our own image we need create dockerfile with a simple syntax for defining steps the steps needed to create an image and run it. Each instruction in dockerfile creates a layer in the container, when we change the dockerfile and rebuild it then only those layers which have been changed are rebuilt.

**4.2.4) Ansible Playbook**

Playbooks are Ansible’s configuration, deployment, and orchestration language. They can describe a policy you want your remote systems to enforce. Playbook can be used to manage configurations of and deployments to remote machines.

**4.2.5) Docker Images**

An image is an inert, immutable, file that's essentially a snapshot of a container. Images are created with the build command, and they'll produce a container when started with run. Images are stored in a Docker registry such as registry.hub.docker.com. Because they can become quite large, images are designed to be composed of layers of other images, allowing a minimal amount of data to be sent when transferring images over the network.

**4.2.6) Docker Container**

A container is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings. Containers are an abstraction at the app layer that packages code and dependencies together. Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space. Containers take up less space than VMs

**4.2.7) Docker daemon**

The Docker daemon listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.

**CHAPTER 5**

**IMPLEMENTATION**

5.1) Packages Used Description

1. docker-py

A Python library for the Docker Engine API. It lets you do anything the docker command does, but from within Python apps – run containers, manage containers, manage Swarms, etc. The package can be installed using the following commands on Ubuntu:

pip install docker

1. docker-compose

Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application’s services. Then, with a single command, you create and start all the services from your configuration.

pip install docker-compose

**5.2) Steps carried out in the Project Implementation**

**5.2.1) Implementation on Master Computer**

1. Updated apt-get libraries on the Master computer.

$ sudo apt-get update

1. Installed Docker CE 17.03 package from “<https://download.docker.com/linux/ubuntu/dists/xenial/pool/stable/amd64/>” using the following command,

$ sudo dpkg -i /path/to/package.deb

1. Create a group for docker which grant root permission and added the user to the group

$ sudo groupadd docker

$ sudo usermod -aG docker $USER

1. Change Proxy settings for Docker so that we can use it behind the proxy firewall,
2. First, Create a systemd drop-in directory for the docker service:

$ mkdir -p /etc/systemd/system/docker.service.d

1. Create a file called /etc/systemd/system/docker.service.d/http-proxy.conf that adds the HTTP\_PROXY environment variable:

[Service]

Environment="HTTP\_PROXY=172.16.19.10:80/"

1. Create a file called /etc/systemd/system/docker.service.d/https-proxy.conf that adds the HTTPS\_PROXY environment variable:

[Service]

Environment="HTTPS\_PROXY=172.16.19.10:80/"

1. Flush changes & Restart Docker:

$ sudo systemctl daemon-reload

$ sudo systemctl restart docker

1. Installed OpenSSH server and enabled SSH.

$ sudo apt-get install openssh-server

1. Installed Ansible via apt-get

$ sudo apt-get update  
$ sudo apt-get install software-properties-common  
$ sudo apt-add-repository ppa:ansible/ansible  
$ sudo apt-get update  
$ sudo apt-get install ansible

1. Installed latest version of ‘docker-py’ and ‘docker-compose’ using pip install.

$ sudo pip install docker-py  
$ sudo pip install docker-compose

1. Add host computer information to the Ansible hosts(inventory) file,

[lab]  
172.16.51.73 ansible\_user=test ansible\_sudo\_pass=test ansible\_ssh\_pass=test

1. Create a dockerfile for MySQL server and save it in a separate directory
2. Create a dockerfile for nginx server and save it in a directory “site” containing the html page and nginx configuration file.
3. Write an Ansible playbook to copy the dockerfile to the host computer, build the docker image and run the docker container.

**5.2.2) Implementation on Host Computer**

1. Create a “test” user with root permissions and set the password.

$ sudo adduser test  
$ usermod -aG sudo test

1. Updated apt-get libraries on the host computer.

$ sudo apt-get update

1. Installed Docker CE 17.03 package from “<https://download.docker.com/linux/ubuntu/dists/xenial/pool/stable/amd64/>” using the following command,

$ sudo dpkg -i /path/to/package.deb

1. Create a group for docker which grant root permission and added the user to the group

$ sudo groupadd docker

$ sudo usermod -aG docker $USER

1. Change Proxy settings for Docker so that we can use it behind the proxy firewall,
2. First, Create a systemd drop-in directory for the docker service:

$ mkdir -p /etc/systemd/system/docker.service.d

1. Create a file called /etc/systemd/system/docker.service.d/http-proxy.conf that adds the HTTP\_PROXY environment variable:

[Service]

Environment="HTTP\_PROXY=172.16.19.10:80/"

1. Create a file called /etc/systemd/system/docker.service.d/https-proxy.conf that adds the HTTPS\_PROXY environment variable:

[Service]

Environment="HTTPS\_PROXY=172.16.19.10:80/"

1. Flush changes & Restart Docker:

$ sudo systemctl daemon-reload

$ sudo systemctl restart docker

1. Installed OpenSSH server and enabled SSH.

$ sudo apt-get install openssh-server

1. Once the image is built and container is created using ansible, we can run the nginx and MySQL containers

$ docker run -it <imagename or image id>

**CHAPTER 6**

**TEST DOCUMENTATION**

* 1. MySQL Server Test

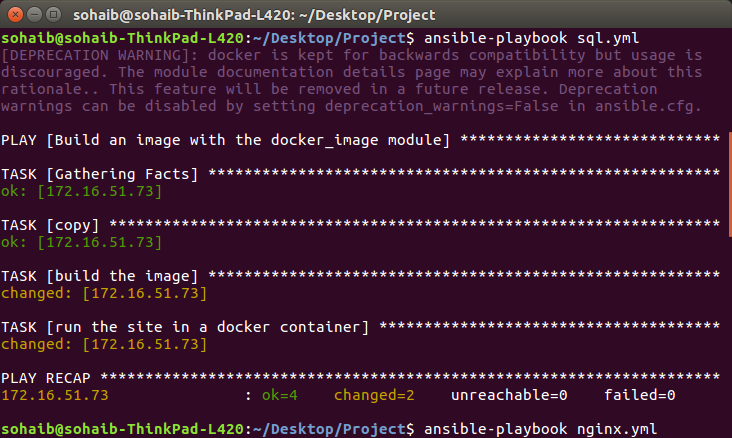


Figure 6.1: Successful Execution of Ansible Playbook sql.yml

The master computer run Ansible playbook which complete a set of commands. It first connects to the host computer via SSH and then copies the dockerfile to the host computer. It then builds the docker image and runs the docker container. The above screenshot shows successfully execution of ansible playbook.

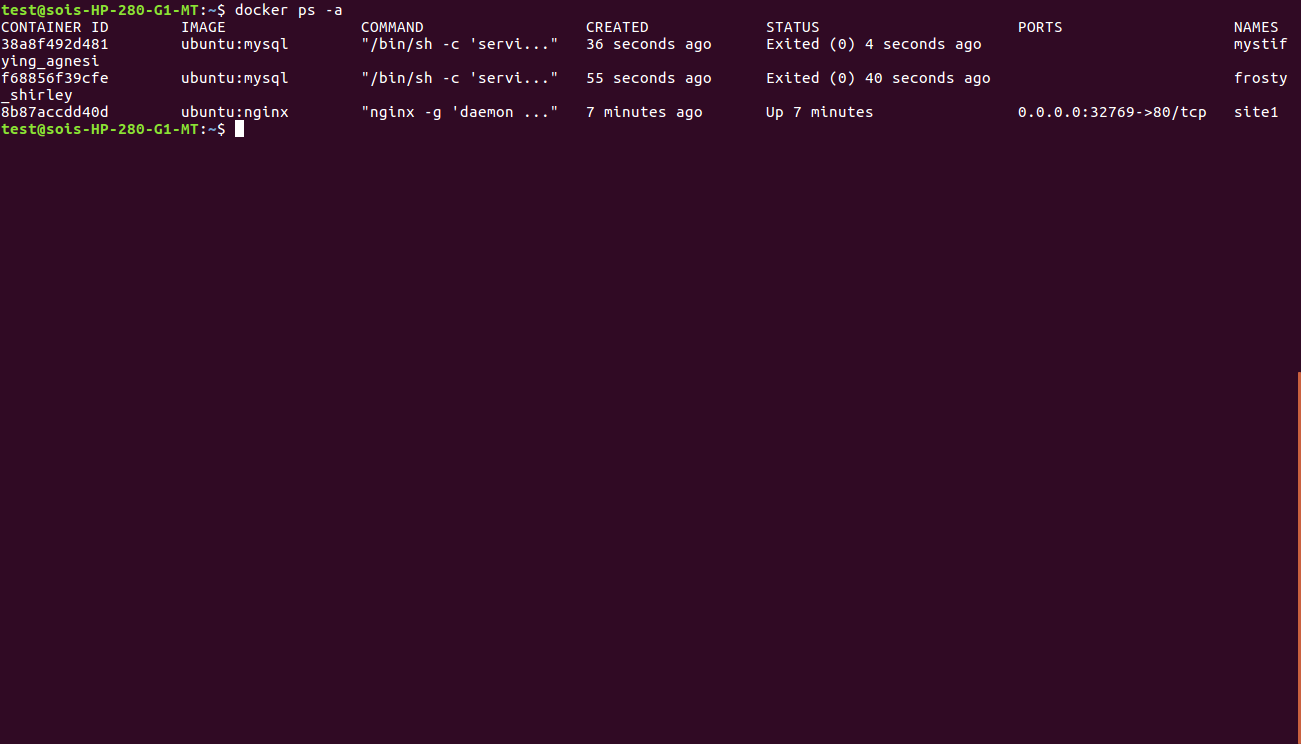


Figure 6.2: Docker containers running on the host machine

Using the below command, we can display the current container running

$docker ps  
  
To get the history of containers executed, we can use the below command (as shown in the above screenshot)

$docker ps -a

Now we can run the docker container using the command,

$ docker run -it <image name or image id>

After executing the command, we can access the container having the MySQL server, by logging into the server using the username and password specified in the dockerfile.

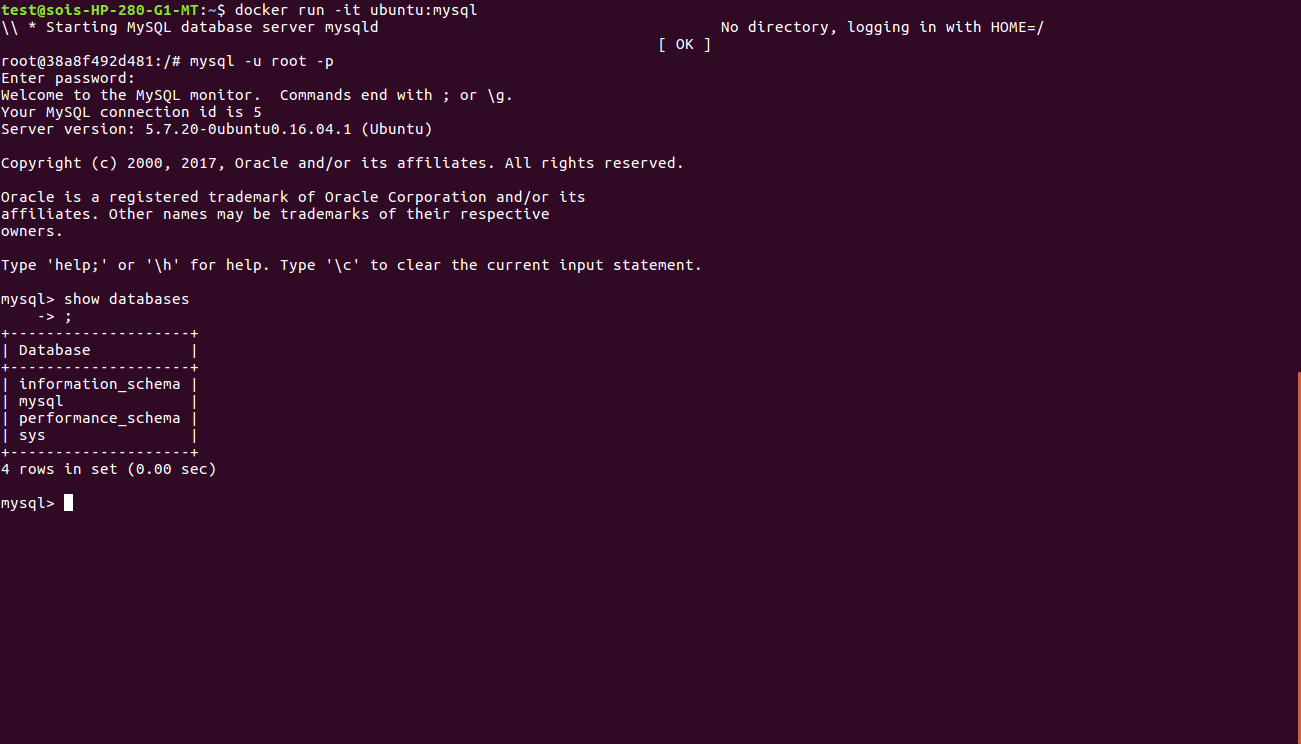


Figure 6.3: Docker Container running MySQL server

* 1. Nginx Server

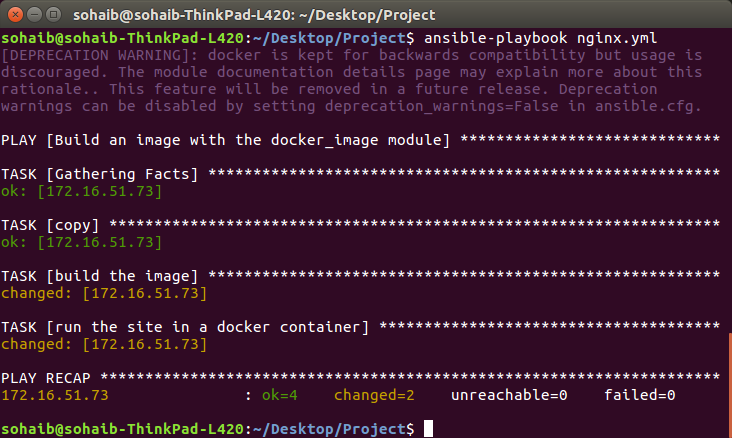


Figure 6.4: Successful Execution of Ansible Playbook nginx.yml

The master computer run Ansible playbook which complete a set of commands. It first connects to the host computer via SSH and then copies the dockerfile to the host computer. It then builds the docker image and runs the docker container. The above screenshot shows successfully execution of ansible playbook.

We can run the nginx docker container with the command,

$ docker run -it ubuntu:nginx

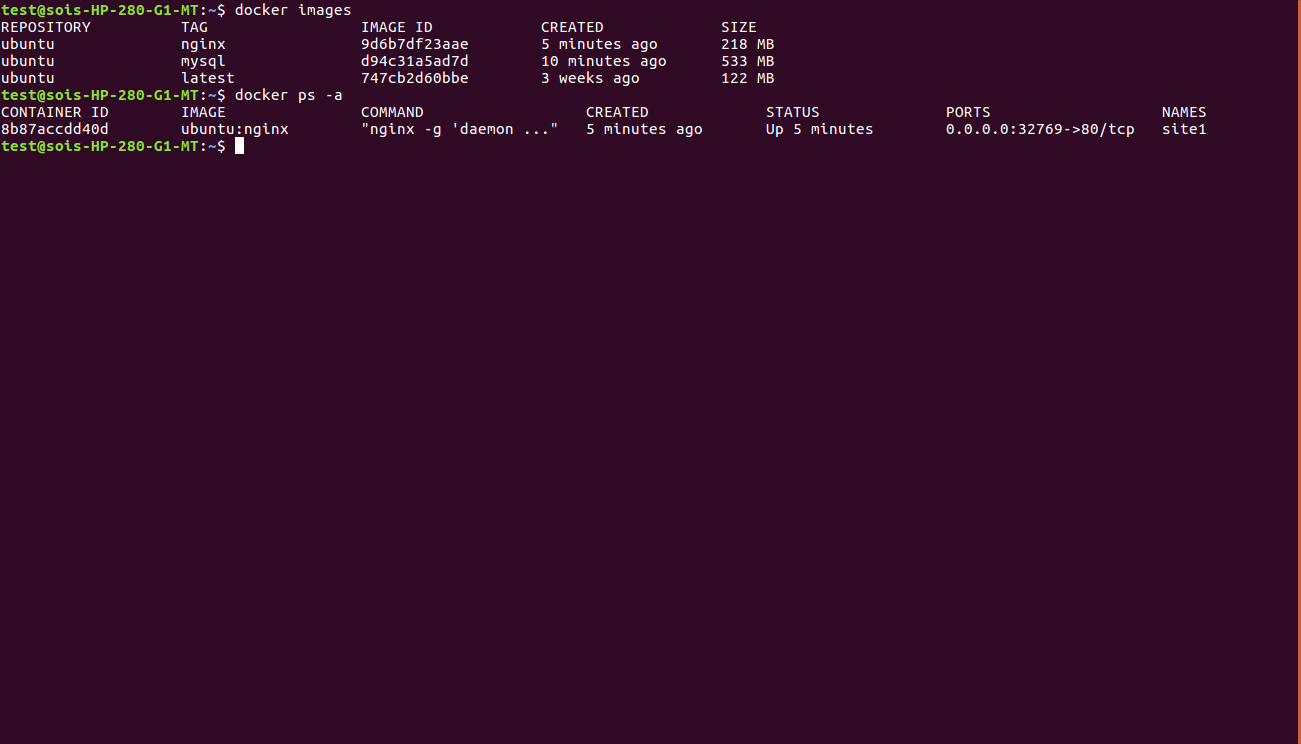


Figure 6.5: Docker containers running on the host machine

We can access the html page using localhost:portnumber on the host machine

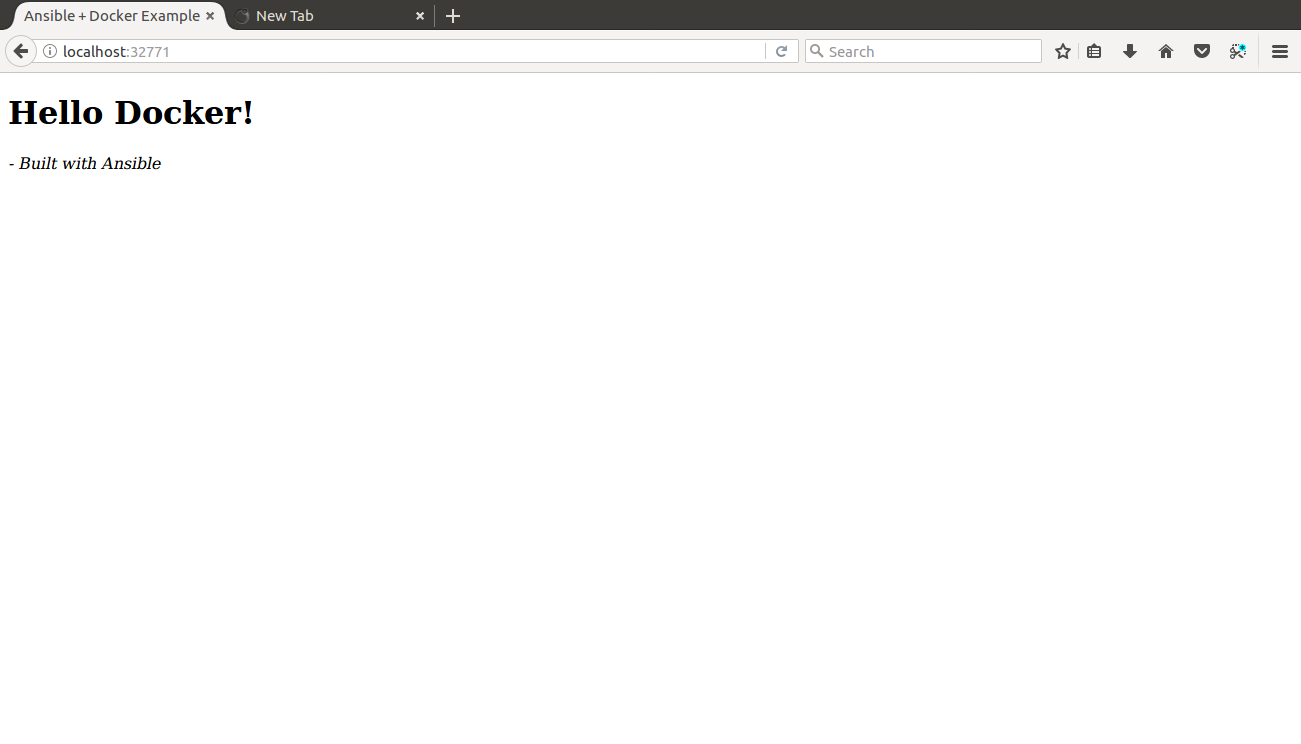
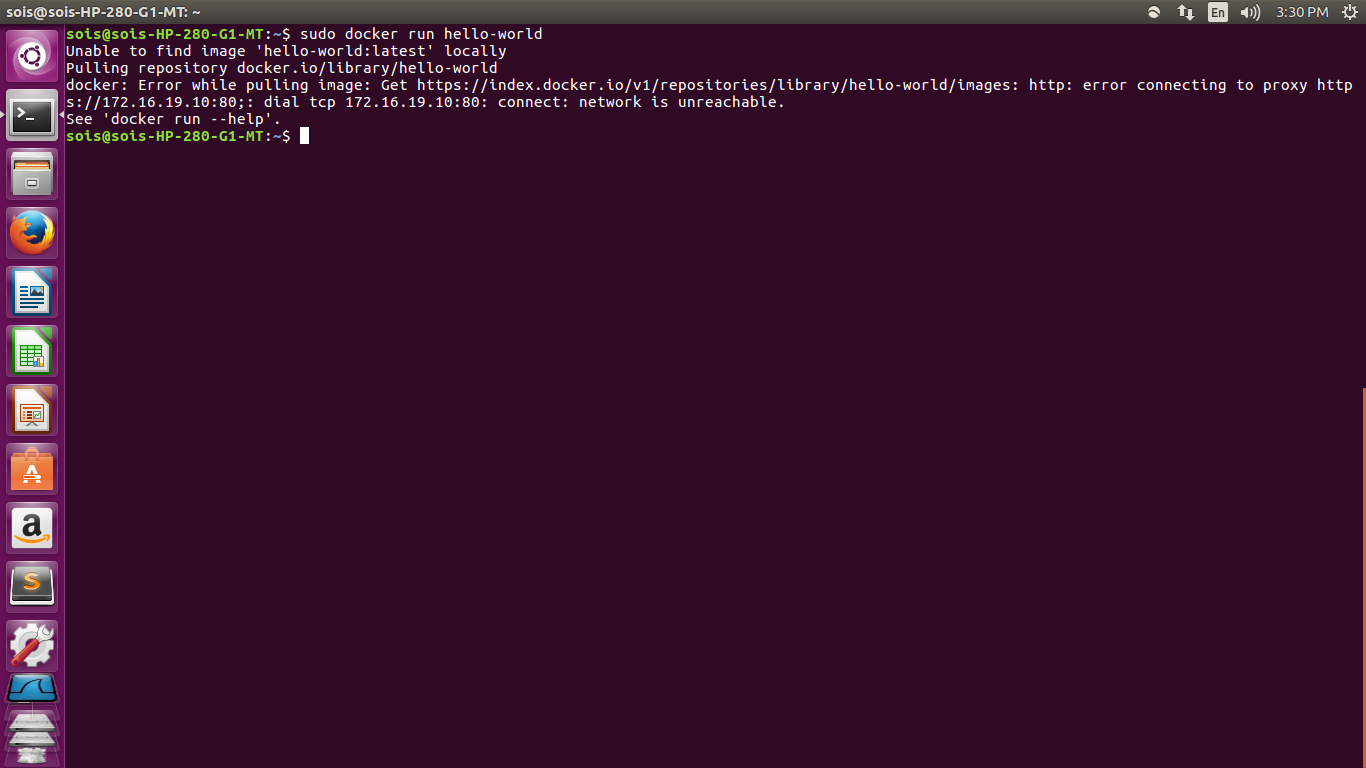


Figure 6.6: Webpage accessed through the nginx server

* 1. Proxy Test

If the host computer isn’t configured with the Proxy setting, Docker will be unable to pull images from the repository. The below screenshot clearly exhibits the error occurred when using Docker in a proxy environment.

Figure 6.7: Failed Proxy Test

**CHAPTER 7**

**REFERENCES**

**7.1 – Bibliography**

[1]. Michael Heap, Ansible - From Beginner to Pro, NY: Springer Science + Business Media, 2016

**7.2 - Webliography**

[1]. C. Anderson, "Docker [Software engineering]," in IEEE Software, May-June 2015, vol. 32, no. 3, pp. 102-c3.

[2]. T. Inagaki, Y. Ueda and M. Ohara, "Container management as emerging workload for operating systems," 2016 IEEE International Symposium on Workload Characterization (IISWC), Providence, RI, 2016, pp. 1-10.

[3]. Red Hat. (2017). *Ansible In Depth* [Online]. Available: http://cdn2.hubspot.net/hub/330046/file-480366556-pdf/pdf\_content/Ansible\_in\_Depth.pdf?t=1390852822000

**APPENDIX**

1. **Source Code**

**A.1) Dockerfile for MySQL server**

FROM ubuntu  
#Setting proxy  
ENV http\_proxy http://172.16.19.10:80/  
ENV https\_proxy https://172.16.19.10:80/  
RUN apt-get update

# MySQL  
ENV MYSQL\_PWD Pwd123  
RUN echo "mysql-server mysql-server/root\_password password $MYSQL\_PWD" | debconf-set-selections  
RUN echo "mysql-server mysql-server/root\_password\_again password $MYSQL\_PWD" | debconf-set-selections  
RUN apt-get -y install mysql-server  
ENTRYPOINT service mysql start && /bin/bash  
EXPOSE 3306

**A.2) Dockerfile for nginx server**

FROM ubuntu:latest

#Setting proxy  
ENV http\_proxy http://172.16.19.10:80/  
ENV https\_proxy https://172.16.19.10:80/  
RUN apt-get update  
RUN apt-get install -y nginx   
ADD nginx.conf /etc/nginx/nginx.conf  
RUN mkdir -p /var/www/site  
ADD index.html /var/www/site/index.html  
EXPOSE 80  
CMD ["nginx", "-g", "daemon off;"]

**A.3) Playbook for MySQL server**

---

- name: Build an image with the docker\_image module

hosts: lab

tasks:

- copy:

src: mysql

dest: /tmp

mode: 0644

- name: build the image

docker\_image: >

name=ubuntu

tag=mysql

path=/tmp/mysql

state=present

- name: run the site in a docker container

docker:

name: site1

image: "ubuntu:mysql"

state: reloaded

publish\_all\_ports: yes

**A.4) Playbook for nginx server**

---

- name: Build an image with the docker\_image module

hosts: lab

tasks:

- copy:

src: site

dest: /tmp

mode: 0644

- name: build the image

docker\_image: >

name=ubuntu

tag=nginx

path=/tmp/site

state=present

- name: run the site in a docker container

docker:

name: site1

image: "ubuntu:nginx"

state: reloaded

publish\_all\_ports: yes

**A.5) Index HTML**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>Ansible + Docker Example</title>

</head>

<body>

<h1>Hello Docker!</h1>

<p><em>- Built with Ansible</em></p>

</body>

</html>

**A.6) Nginx configuration file**

user www-data;

worker\_processes 1;

events {

worker\_connections 1024;

}

http {

server {

listen 80;

root /var/www/site;

}

}