	Hope Foundation's **Finolex Academy of Management and Technology, Ratnagiri** **Information Technology Department**

Subject name: DevOps Lab			Subject Code: ITL803
Class	BE IT	Semester – VIII (CBCGS)	Academic year: 2019-20
Name of Student		**QUIZ Score :**	
Roll No		Assignment/Experiment No.	06

Title: Build, deploy and manage python based web application on Docker.

1.Lab objectives applicable

LOB5. To use Docker to deploy and manage Software applications running on Container

2. Lab outcomes applicable:

LO4-Analyze & Illustrate the Containerization of OS images and deployment of applications over Docker

3. Learning Objectives:

1. Understand the deployment of applications on container
2. Understand the launching of an app on containers.

4. Practical applications of the assignment/experiment: To automate the several tasks such as automatic building the code ,deploying the code and notifying the developer about build status via sms/email etc

5. Prerequisites:

1. Familiar with Linux os
2. Internet Access
3. Nginx installed

6. Hardware Requirements:

1. Internet Access with Browser
2. Access to root privileges on Ubuntu 18.04

7. Software Requirements:

Web server(Nginx)

8. Quiz Questions (if any): (Online Exam will be taken separately batchwise, attach the certificate/ Marks obtained)

1. What is difference between docker image and docker container
2. How to get docker images
3. What are the benefits of containers?

9. Experiment/Assignment Evaluation:

Sr. No.	**Parameters**	**Marks obtained**	**Out of**
1	Technical Understanding (Assessment may be done based on Q & A **or** any other relevant method.) Teacher should mention the other method used -		6
2	Neatness/presentation		2
3	Punctuality		2

Date of performance (DOP)		**Total marks obtained**		**10**
Date of checking (DOC)		**Signature of teacher**		

10.Theory-. **<Preferably given as handwritten work for students>**

Docker is an open-source application that allows administrators to create, manage, deploy, and replicate applications using containers. Containers can be thought of as a package that houses dependencies that an application requires to run at an operating system level. This means that each application deployed using Docker lives in an environment of its own and its requirements are handled separately.

Flask is a web micro-framework that is built on Python. It is called a micro-framework because it does not require specific tools or plug-ins to run. The Flask framework is lightweight and flexible, yet highly structured, making it preferred over other frameworks.

Deploying a Flask application with Docker will allow you to replicate the application across different servers with minimal reconfiguration.

In this tutorial, you will create a Flask application and deploy it with Docker. This tutorial will also cover how to update an application after deployment.

11. Installation Steps / Performance Steps –

```
sudo mkdir /var/www/TestApp
```

Move in to the newly created TestApp directory:

```
cd /var/www/TestApp
```

Next, create the base folder structure for the Flask application:

```
sudo mkdir -p app/static app/templates
```

Run the following command to create the file:

```
sudo nano app/__init__.py
```

Packages in Python allow you to group modules into logical namespaces or hierarchies. This approach enables the code to be broken down into individual and manageable blocks that perform specific functions.

Next, you will add code to the __init__.py that will create a Flask instance and import the logic from the views.py file, which you will create after saving this file. Add the following code to your new file:

/var/www/TestApp/app/__init__.py

```
from flask import Flask
app = Flask(__name__)
from app import views
```

Once you've added that code, save and close the file.

With the __init__.py file created, you're ready to create the views.py file in your app directory. This file will contain most of your application logic.

```
sudo nano app/views.py
```

Next, add the code to your views.py file. This code will return the hello world! string to users who visit your web page:

/var/www/TestApp/app/views.py

```
from app import app

@app.route('/')
def home():
   return "hello world!"
```

With the views.py file in place, you're ready to create the uwsgi.ini file. This file will contain the *uWSGI* configurations for our application. uWSGI is a deployment option for Nginx that is both a protocol and an application server; the application server can serve uWSGI, FastCGI, and HTTP protocols.

To create this file, run the following command:

```
sudo nano uwsgi.ini
```

Next, add the following content to your file to configure the uWSGI server:

/var/www/TestApp/uwsgi.ini

```
[uwsgi]
module = main
callable = app
master = true
```

Next, copy and paste the following into the file. This imports the Flask instance named app from the application package that was previously created.

/var/www/TestApp/main.py

```
from app import app
```

Finally, create a requirements.txt file to specify the dependencies that the pip package manager will install to your Docker deployment:

```
sudo nano requirements.txt
```

Add the following line to add Flask as a dependency:

/var/www/TestApp/requirements.txt

```
Flask==1.0.2
```

This specifies the version of Flask to be installed. At the time of writing this tutorial, 1.0.2 is the latest Flask version. You can check for updates at the official website for Flask.

Save and close the file. You have successfully set up your Flask application and are ready to set up Docker.

First, create the Dockerfile.

```
sudo nano Dockerfile
```

Next, add your desired configuration to the Dockerfile. These commands specify how the image will be built, and what extra requirements will be included.

/var/www/TestApp/Dockerfile

```
FROM tiangolo/uwsgi-nginx-flask:python3.6-alpine3.7
RUN apk --update add bash nano
ENV STATIC_URL /static
ENV STATIC_PATH /var/www/app/static
COPY ./requirements.txt /var/www/requirements.txt
RUN pip install -r /var/www/requirements.txt
```

```
sudo nc localhost 56733 < /dev/null; echo $?
```

Once you've found an open port to use, create the start.sh script:

```
sudo nano start.sh
```

The start.sh script is a shell script that will build an image from the Dockerfile and create a container from the resulting Docker image. Add your configuration to the new file:

/var/www/TestApp/start.sh

```
#!/bin/bash
app="docker.test"
docker build -t ${app} .
docker run -d -p 56733:80 \
  --name=${app} \
  -v $PWD:/app ${app}
```

Execute the start.sh script to create the Docker image and build a container from the resulting image:

```
sudo bash start.sh
```

Once the script finishes running, use the following command to list all running containers:

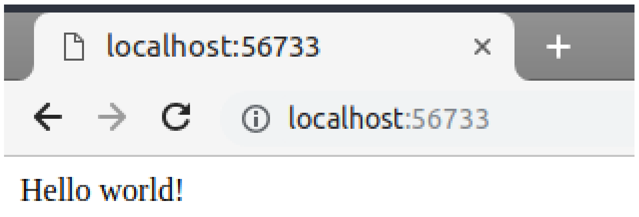
```
sudo docker ps
```

You will receive output that shows the containers:

```
Output
CONTAINER ID       IMAGE            COMMAND              CREATED         STATUS          PORTS
NAMES
58b05508f4dd       docker.test       "/entrypoint.sh /sta…"   12 seconds ago     Up 3 seconds
443/tcp, 0.0.0.0:56733->80/tcp   docker.test
```

You will find that the docker.test container is running. Now that it is running, visit the IP address at the specified port in your browser: http://ip-address:56733

You'll see a page similar to the following:



Hello world!

12. Learning Outcomes Achieved.
1.Student understood the installations of various packages on docker containers
2.Students understood the setting up docker
3.Students understood the launching of app

13. Conclusion:

1. Applications of the studied technique in industry
    a. To reduce the complexities of software plugins and their dependencies
    b. To achieve Plug and Play policy
2. Engineering Relevance
    a. To bypass installtions and configurations
3. Skills Developed
    a. Making containers ready to use

14.References:

1.https://www.wintellect.com/containerize-python-app-5-minutes/
2.https://www.fullstackpython.com/docker.html