VOTING APPLICATION DEPLOYMENT USING DOCKERS

A PROJECT REPORT

Submitted by

Abhinav Anand- 15BCE0060

Course Code: CSE4011

Course Title: Virtualization

In partial fulfilment for the award of the degree of

B.Tech

in

Computer Science and Engineering

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

INDEX

1. **Introduction**

2. **Literature Survey**

3. **Overview of the Work**
   3.1.Problem description
   3.2.Software Requirements
   3.3.Hardware Requirements
   3.4.Network Requirements

4. **System Design**
   4.1. Architecture

5. **Implementation**
   5.1.Description of Modules/Programs
   5.2.Source Code
   5.3.Execution snapshots

6. **Conclusion and Future Directions**
7. **References**

1. Introduction

A simple distributed voting application running across multiple Docker containers. Used to register and tabulate votes to either aid or take care of casting and counting votes. The voting application only accepts one vote per client. It does not register votes if a vote has already been submitted from a client.

Virtualization overview:

Virtualization refers to the act of creating a virtual (rather than actual) version of something, including virtual computer hardware platforms, storage devices, and computer network resources.

It is used for developing, shipping, and running applications. It enables you to separate your applications from your infrastructure so you can deliver software quickly. You can manage your infrastructure in the same ways you manage your applications. By taking advantage of these methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

The Docker platform

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allow you to run many containers simultaneously on a given host. Containers are lightweight because they don’t need the extra load of a hypervisor but run directly within the host machine’s kernel. This means you can run more containers on a given hardware combination than if you were using virtual machines. You can even run Docker containers within host machines that are actually virtual machines!

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

- Develop your application and its supporting components using containers.
- The container becomes the unit for distributing and testing your application.

- When you’re ready, deploy your application into your production environment, as a container or an orchestrated service. This works the same whether your production environment is a local data center, a cloud provider, or a hybrid of the two.

What can I use Docker for?

Fast, consistent delivery of your applications

Docker streamlines the development lifecycle by allowing developers to work in standardized environments using local containers which provide your applications and services. Containers are great for continuous integration and continuous delivery (CI/CD) workflows.

Responsive deployment and scaling

Docker’s container-based platform allows for highly portable workloads. Docker containers can run on a developer’s local laptop, on physical or virtual machines in a data center, on cloud providers, 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, so you can use more of your compute capacity to achieve your business goals. Docker is perfect for high density environments and for small and medium deployments where you need to do more with fewer resources.

2. Literature Survey

a. **Title:** An Introduction to Docker and Analysis of its Performance
Year: 2017
Author: Babak Bashari Rad, Harrison John Bhatti, Mohammad Ahmadi
Description: Docker provides some facilities, which are useful for developers and administrators. It is an open platform can be used for building, distributing, and running applications in a portable, lightweight runtime and packaging tool, known as Docker Engine. It also provides Docker Hub, which is a cloud service for sharing applications. Costs can be reduced by replacing traditional virtual machine with docker container. It excellently reduces the cost of re-building the cloud development platform. Docker automates the applications when they are containerized. An extra layer of docker engine is added to the host operating system. The performance of docker is faster than virtual machines as it has no guest operating system and less resource overhead

b. **Title:** Analysis of Docker Security
Year: 2015
Author: Thanh Bui
Description: Container-based virtualization and hypervisor-based virtualization are two main types of virtualization technolo-
gies that have emerged to the market. Of these two classes, container-based virtualization is able to provide a more lightweight and efficient virtual environment, but not without security concerns. In this paper, we analyze the security level of Docker, a well-known representative of container-based approachesThe analysis considers two areas: (1) the inter-
nal security of Docker, and (2) how Docker interacts with the security features of the Linux kernel, such as SELinux and AppArmor, in order to harden the host system. Furthermore, the paper also discusses and identifies what could be done when using Docker to increase its level of security

c. **Title:** Research and implementation of Docker performance service in distributed platform
Year: 2015
Author: Liu Lijuan
Description: This paper takes the popular Docker technology as the starting point, in order to meet the needs of the development of container cloud, the performance and service of Docker in the distributed platform are studied. It solves the shortcomings of the current Docker container only for single node application, and expects further research on the Docker platform in the future, to improve the development 、 deployment、operation and maintenance efficiency of the Docker platform.

3. Overview of the Work

a. Problem description

A simple distributed voting application running across multiple Docker containers. Used to register and tabulate votes to either aid or take care of casting and counting votes. The voting application only accepts one vote per client. It does not register votes if a vote has already been submitted from a client.

b. Software Requirements

Download Docker Desktop for Mac or Windows. Docker Compose will be automatically installed. On Linux, make sure you have the latest version of Compose.

Linux container:

The Linux stack uses Python, Node.js, .NET Core (or optionally Java), with Redis for messaging and Postgres for storage.

If you're using Docker Desktop on Windows, you can run the Linux version by switching to Linux containers, or run the Windows containers version.

Run in this directory: docker-compose up

The app will be running at http://localhost:5000, and the results will be at http://localhost:5001.

Alternately, if you want to run it on a Docker Swarm, first make sure you have a swarm. If you don't, run:

```
docker swarm init
```

Once you have your swarm, in this directory run:

```
docker stack deploy --compose-file docker-stack.yml vote
```

Windows container:

An alternative version of the app uses Windows containers based on Nano Server. This stack runs on .NET Core, using NATSfor messaging and TiDB for storage.

You can build from source using:

```
docker-compose -f docker-compose-windows.yml build
```

Then run the app using:

```
docker-compose -f docker-compose-windows.yml up -d
```

Or in a Windows swarm, run docker stack deploy -c docker-stack-windows.yml vote
The app will be running at http://localhost:5000, and the results will be at http://localhost:5001.

c. Hardware Requirements

1. CS Docker Engine 1.13, or EE Daemon 17.03 and higher
2. 8.00 GB of RAM for manager nodes or nodes running DTR
3. 4.00 GB of RAM for worker nodes
4. 3.00GB of available disk space

d. Network requirements

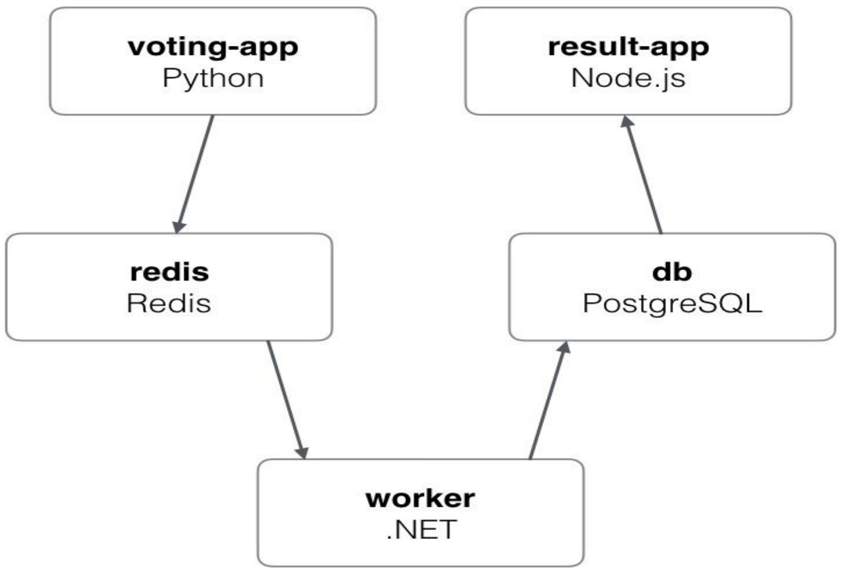
When installing UCP on a host, make sure the following ports are open:

Hosts	Direction	Port	Purpose
managers, workers	in	TCP 443 (configurable)	Port for the UCP web UI and API
managers	in	TCP 2376 (configurable)	Port for the Docker Swarm manager. Used for backwards compatibility
managers, workers	in	TCP 2377 (configurable)	Port for communication between swarm nodes
managers, workers	in, out	TCP, UDP 4789	Port for overlay networking
managers, workers	in, out	TCP, UDP 7946	Port for gossip-based clustering
managers, workers	in	TCP 12376	Port for a TLS proxy that provides access to UCP, Docker Engine, and Docker Swarm
managers	in	TCP 12379	Port for internal node configuration, cluster configuration, and HA

Hosts	Direction	Port	Purpose
managers	in	TCP 12380	Port for internal node configuration, cluster configuration, and HA
managers	in	TCP 12381	Port for the certificate authority
managers	in	TCP 12382	Port for the UCP certificate authority
managers	in	TCP 12383	Port for the authentication storage backend
managers	in	TCP 12384	Port for the authentication storage backend for replication across managers
managers	in	TCP 12385	Port for the authentication service API
managers	in	TCP 12386	Port for the authentication worker
managers	in	TCP 12387	Port for the metrics service

4. System Design:

4.1 Architecture:



- A front-end web app in Python or ASP.NET Core which lets you vote between two options
- A Redis or NATS queue which collects new votes
- A .NET Core, Java or .NET Core 2.1 worker which consumes votes and stores them in…
- A Postgres or TiDB database backed by a Docker volume
- A Node.js or ASP.NET Core SignalR webapp which shows the results of the voting in real time

The voting application only accepts one vote per client. It does not register votes if a vote has already been submitted from a client.

5. Implementation

5.1 Description of Modules/Programs

Voting-app

The main application which takes in the votes and sends it to redis which is an nosql database and stores it there to allow different application to then use it. It is a combination of a python code to send the values to the database, a html and css script to create a pleasant webpage which can be used by the user

Worker

Worker is a thread based program which consumes votes and stores them in the required database. Each voting application has its associated thread included with it and dies when it is closed

Result-app

Result application is the part of the container which showcases the results of all the votes put into the database. The results are all shown on a webpage which takes the result from the database and show it to the users

5.2 Source Code

App.py

```
from flask import Flask, render_template, request, make_response, g
from redis import Redis
import os
import socket
import random
import json

option_a = os.getenv('OPTION_A', "Cats")
option_b = os.getenv('OPTION_B', "Dogs")
hostname = socket.gethostname()

app = Flask(__name__)

def get_redis():
    if not hasattr(g, 'redis'):
        g.redis = Redis(host="redis", db=0, socket_timeout=5)
    return g.redis

@app.route("/", methods=['POST','GET'])
def hello():
    voter_id = request.cookies.get('voter_id')
    if not voter_id:
        voter_id = hex(random.getrandbits(64))[2:-1]

    vote = None

    if request.method == 'POST':
        redis = get_redis()
        vote = request.form['vote']
        data = json.dumps({'voter_id': voter_id, 'vote': vote})
        redis.rpush('votes', data)

    resp = make_response(render_template(
        'index.html',
        option_a=option_a,
        option_b=option_b,
        hostname=hostname,
        vote=vote,
    ))
    resp.set_cookie('voter_id', voter_id)
    return resp


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

Main Docker-compose

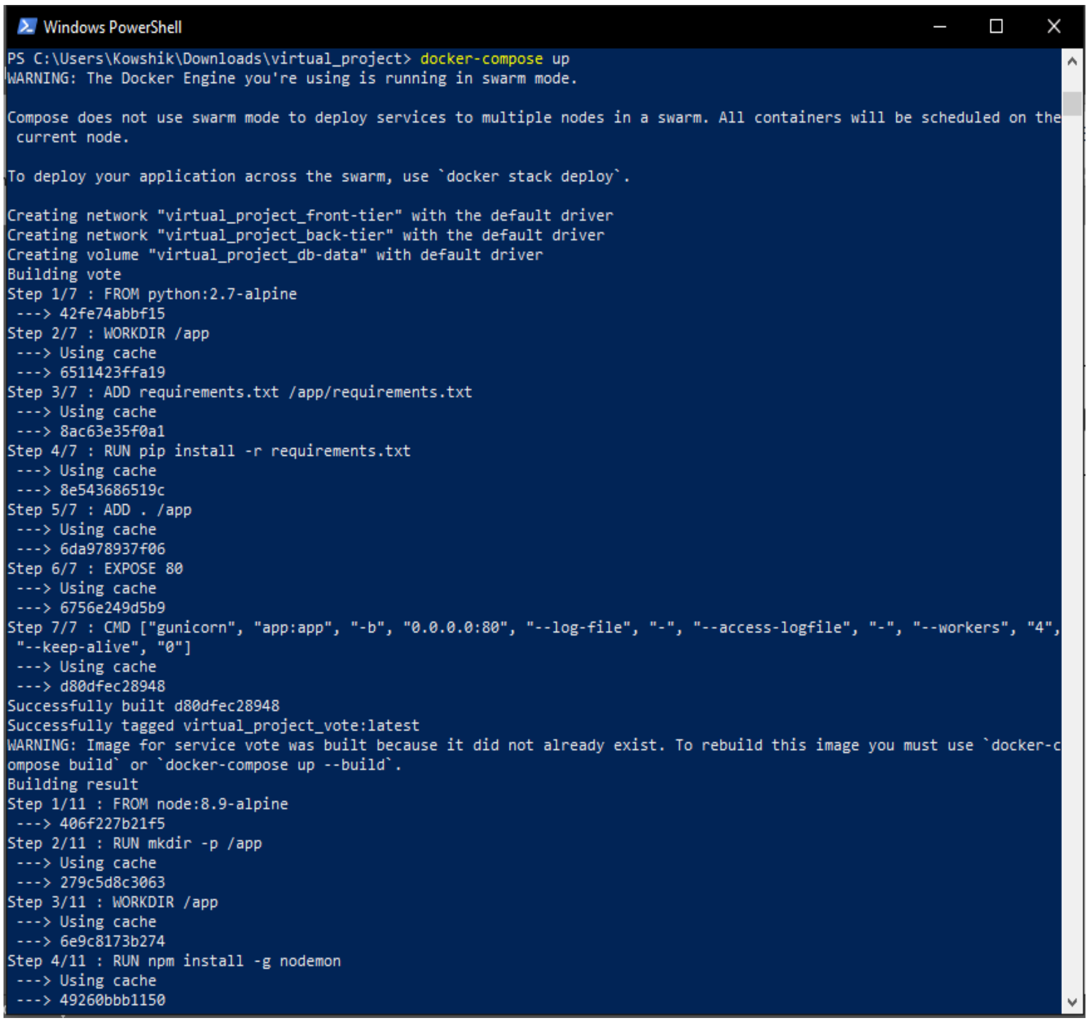
```
version: "3"

services:
  vote:
    build: ./vote
    command: python app.py
    volumes:
     - ./vote:/app
    ports:
      - "5000:80"
    networks:
      - front-tier
      - back-tier

  result:
    build: ./result
    command: nodemon server.js
    volumes:
      - ./result:/app
    ports:
      - "5001:80"
      - "5858:5858"
    networks:
      - front-tier
      - back-tier

  worker:
    build:
      context: ./worker
    depends_on:
      - "redis"
    networks:
      - back-tier

  redis:
    image: redis:alpine
    container_name: redis
    ports: ["6379"]
    networks:
      - back-tier

  db:
    image: postgres:9.4
    container_name: db
    volumes:
      - "db-data:/var/lib/postgresql/data"
    networks:
      - back-tier

volumes:
  db-data:

networks:
  front-tier:
  back-tier:
```

Vote dockerfile

```
# Using official python runtime base image
FROM python:2.7-alpine

# Set the application directory
WORKDIR /app

# Install our requirements.txt
ADD requirements.txt /app/requirements.txt
RUN pip install -r requirements.txt

# Copy our code from the current folder to /app inside the container
ADD . /app

# Make port 80 available for links and/or publish
EXPOSE 80

# Define our command to be run when launching the container
CMD ["gunicorn", "app:app", "-b", "0.0.0.0:80", "--log-file", "-", "--
access-logfile", "-", "--workers", "4", "--keep-alive", "0"]
```

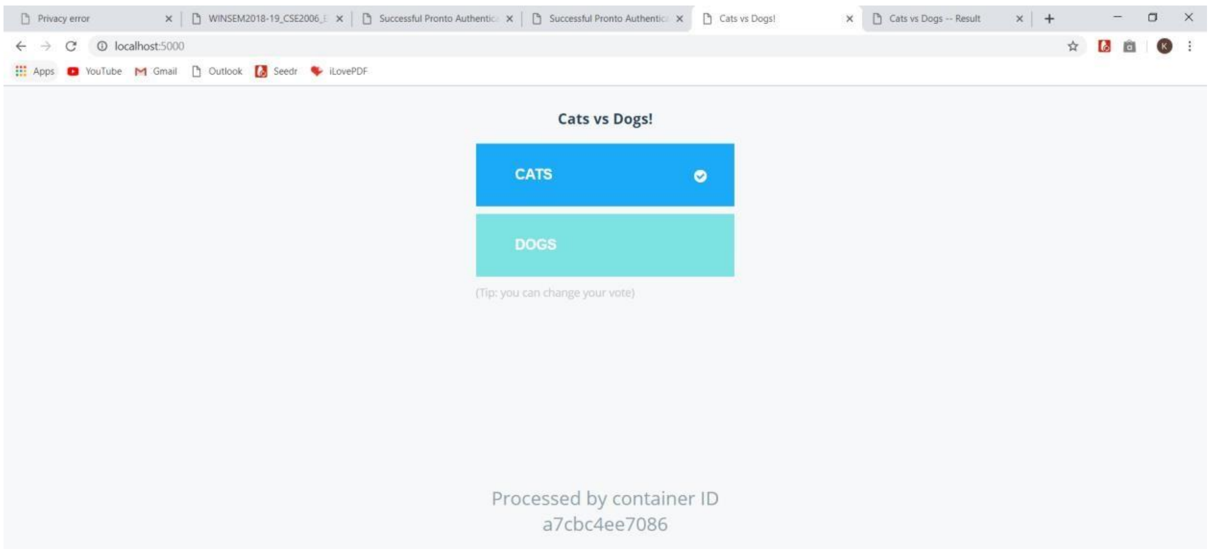
5.3 Execution snapshots

Building the image



```
PS C:\Users\Kowshik\Downloads\virtual_project> docker-compose up
WARNING: The Docker Engine you're using is running in swarm mode.

Compose does not use swarm mode to deploy services to multiple nodes in a swarm. All containers will be scheduled on the
 current node.

To deploy your application across the swarm, use `docker stack deploy`.

Creating network "virtual_project_front-tier" with the default driver
Creating network "virtual_project_back-tier" with the default driver
Creating volume "virtual_project_db-data" with default driver
Building vote
Step 1/7 : FROM python:2.7-alpine
 ---> 42fe74abbf15
Step 2/7 : WORKDIR /app
 ---> Using cache
 ---> 6511423ffa19
Step 3/7 : ADD requirements.txt /app/requirements.txt
 ---> Using cache
 ---> 8ac63e35f0a1
Step 4/7 : RUN pip install -r requirements.txt
 ---> Using cache
 ---> 8e543686519c
Step 5/7 : ADD . /app
 ---> Using cache
 ---> 6da978937f06
Step 6/7 : EXPOSE 80
 ---> Using cache
 ---> 6756e249d5b9
Step 7/7 : CMD ["gunicorn", "app:app", "-b", "0.0.0.0:80", "--log-file", "-", "--access-logfile", "-", "--workers", "4",
 "--keep-alive", "0"]
 ---> Using cache
 ---> d80dfec28948
Successfully built d80dfec28948
Successfully tagged virtual_project_vote:latest
WARNING: Image for service vote was built because it did not already exist. To rebuild this image you must use `docker-c
ompose build` or `docker-compose up --build`.
Building result
Step 1/11 : FROM node:8.9-alpine
 ---> 406f227b21f5
Step 2/11 : RUN mkdir -p /app
 ---> Using cache
 ---> 279c5d8c3063
Step 3/11 : WORKDIR /app
 ---> Using cache
 ---> 6e9c8173b274
Step 4/11 : RUN npm install -g nodemon
 ---> Using cache
 ---> 49260bbb1150
```

Vote Webpage:



Cats vs Dogs!

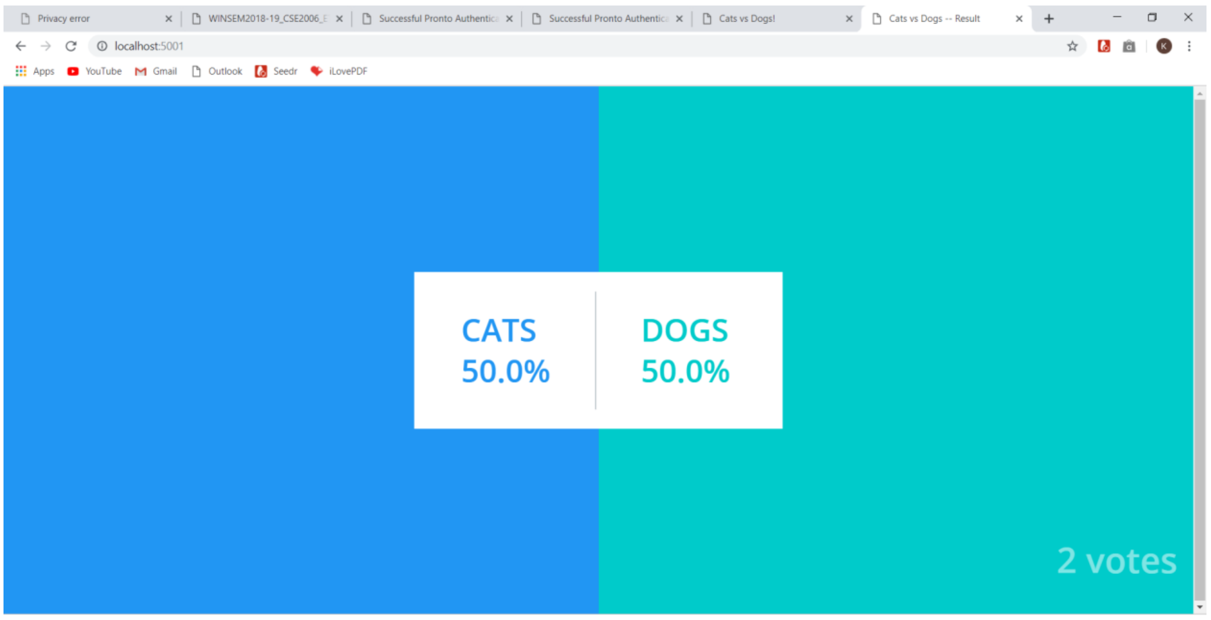
CATS

DOGS

(Tip: you can change your vote)

Processed by container ID
a7cbc4ee7086

Results webpage:



6. Conclusion And Future Work:

In conclusion docker is a completely new field which can be improved significantly and thus be made into something significant. The implementation of this program gives a basic idea on how a portable docker app might be implemented. This for future reference would give a considerable information how a portable and highly secure voting application which is a highly valuable asset in any given scenario.
This also show the versatility of the docker platform due to the wide range of applications and other platforms which can be easily build on it. The images build for a docker can easily be moved to platform where the dependencies might not be present.
In the future, we would like to implement better security because right now there is very little in preventing attackers from manipulating votes. We could implement a proper cloud implementation of the program which would allow multiple users to submit their votes

7. References

- *(PDF) Analysis of Docker Security*. Available from: https://www.researchgate.net/publication/270906436_Analysis_of_Docker_Security [accessed Apr 09 2019].
- http://paper.ijcsns.org/07_book/201703/20170327.pdf
- file:///C:/Users/Bharath/Downloads/2632-Article%20Text-4760-1-10-20180103.pdf
- https://github.com/dockersamples/example-voting-app/wiki
- https://docs.docker.com/v17.09/datacenter/ucp/2.1/guides/admin/install/system-requirements/
- Liu Minxian, Design and implementation of service invocation topology analysis and performance monitoring system based on Docker, Zhejiang: Zhejiang University, 2016.
- Liang Junjie, Research and implementation of cloud resource scheduling based on application container, Sichuan: University of Electronic Science and Technology of China, 2015.
- https://www.linode.com/docs/applications/containers/docker-container-communication/
- https://stackoverflow.com/questions/37800631/how-to-connect-to-server-on-docker-from-host-machine/37823465
- https://docs.docker.com/engine/reference/commandline/docker/