Michael Tse
11/24/19

Application Security – Assignment 4 – Fall 2019

Github Repository - https://github.com/mt1836/Assignment4

INTRODUCTION

In this assignment we containerized our web application from assignment 3 by using Docker. Aside from containerization, we also utilized features in Docker that provides automated builds, replica creation and secrets management through the use of docker-compose, docker stack, docker swarm and docker secrets.

IMPLEMENTATION

The first step in containerizing our web app involved creating a dockerfile which is essentially a text file that tells docker how to build your applications environment with all of the pre-requisites to run your service/application. Below is an image of the dockerfile we used:

2019_Fall_Application_Security > Assignment4 > Dockerfi

```
1   FROM ubuntu:19.10
2   COPY . /Assignment4
3   WORKDIR /Assignment4
4   RUN apt-get update
5   RUN apt-get install -y python3.7
6   RUN apt-get install -y python3-pip
7   RUN pip3 install flask
8   RUN pip3 install bcrypt
9   RUN pip3 install flask_sqlalchemy
10  RUN pip3 install flask_login
11  RUN pip3 install flask_wtf
12  CMD flask run --host=0.0.0.0
```

Once the dockerfile has been created we can build the image by running the following command. In the example below we built an image that gets stored in a local repository called assignment4.

```
root@appsec-VirtualBox:/media/sf_NYU/2019_Fall_Application_Security/Assignment4# docker build -t assignment4 .
```

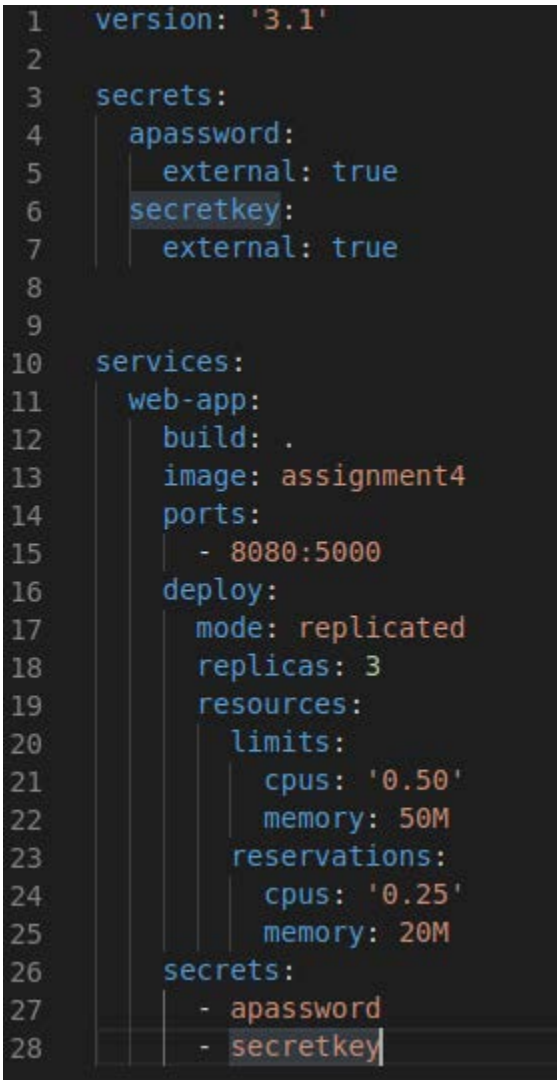
We can see the image in the following screenshot along with a few other images we have for the base Ubuntu OS.

```
root@appsec-VirtualBox:/media/sf_NYU/2019_Fall_Application_Security/Assignment4# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
assignment4         latest              2d3645427379        3 hours ago         600MB
ubuntu              18.04               775349758637        3 weeks ago         64.2MB
ubuntu              latest              775349758637        3 weeks ago         64.2MB
ubuntu              19.10               09604a62a001        5 weeks ago         72.9MB
root@appsec-VirtualBox:/media/sf_NYU/2019_Fall_Application_Security/Assignment4#
```

From here we can perform a proof of concept to ensure that our application works under one container by creating the container using docker create and then starting it using docker start, or you could use

docker run which combines the create/start into one command. Having a containerized application auto built from a dockerfile is a very robust way to deploy code as others can download the dockerfile and be able to build the environment that will run your application without issue very quick and easily.

Being able to do this with one container is great but in more complex deployments where you may have a web server and a db server (we have a combined web and db server in this assignment) you don't want to manually run docker commands to spin up each individual container. This is where docker-compose.yml files come into play. Docker-compose.yml files take the images built by the dockerfile and spin up instances of the images in the form of containers. It can spin up a web server a database server and set port mapping settings as shown in the screenshot below:



```
version: '3.1'

secrets:
  apassword:
    external: true
  secretkey:
    external: true


services:
  web-app:
    build: .
    image: assignment4
    ports:
      - 8080:5000
    deploy:
      mode: replicated
      replicas: 3
      resources:
        limits:
          cpus: '0.50'
          memory: 50M
        reservations:
          cpus: '0.25'
          memory: 20M
    secrets:
      - apassword
      - secretkey
```

Docker-compose up would be the command you use to build based off the dockerfile and docker-compose.yml file with some limitations. Docker-compose does not recognize the deploy and secrets section of the docker-compose.yml file. For this reason we need to utilize docker stack deploy which will execute the docker-compose.yml file in its entirety.

Before running this command however we need to handle secrets. Secrets are a way to securely store sensitive data such as passwords and tokens needed for automatic deployments/builds. By using secrets these sensitive data points can be excluded from plain text view in dockerfiles and our code which is uploaded to a public repository that anyone would be able to see. To setup secrets we need to initialize docker swarm using the following command:

```
root@appsec-VirtualBox:/media/sf_NYU/2019_Fall_Application_Security/Assignment4# docker swarm init
Swarm initialized: current node (wxfoas2p72hlra2vk2dq6yt3z) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-271jgs99c3isg14jk7ri7gunbqctckc2vfd3udzr8wum2920rv-0bdyhweurkxdf3njogqwq6rbv 10.0.2.15:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

root@appsec-VirtualBox:/media/sf_NYU/2019_Fall_Application_Security/Assignment4#
```

Once this is done we can create the secret by having the owner of the web app input it via the command line using the command below:

```
root@appsec-VirtualBox:/media/sf_NYU/2019_Fall_Application_Security/Assignment4# echo adminpassword | docker secret create apassword -
2492fjsfxwx44gjrpv18jgg5x
root@appsec-VirtualBox:/media/sf_NYU/2019_Fall_Application_Security/Assignment4# echo thisisthesupersecretkey | docker secret create secretkey -
b8igila5qs2r3ichjt85nmnwb
root@appsec-VirtualBox:/media/sf_NYU/2019_Fall_Application_Security/Assignment4#
```

We can see the secrets created by typing docker secret ls. In our assignment we created two secrets. One called apassword which stores the admin password “adminpassword” and another that stores our secretkey “thisisthesupersecretkey”.

```
root@appsec-VirtualBox:/media/sf_NYU/2019_Fall_Application_Security/Assignment4# docker secret ls
ID                          NAME                DRIVER              CREATED             UPDATED
2492fjsfxwx44gjrpv18jgg5x   apassword                               43 seconds ago      43 seconds ago
b8igila5qs2r3ichjt85nmnwb   secretkey                               26 seconds ago      26 seconds ago
root@appsec-VirtualBox:/media/sf_NYU/2019_Fall_Application_Security/Assignment4#
```

Now that we have secrets setup we can create our containers/replicas with resources limited for CPU and memory by running the following docker stack deploy command:

```
root@appsec-VirtualBox:/media/sf_NYU/2019_Fall_Application_Security/Assignment4# docker stack deploy --compose-file docker-compose.yml assignment4
Ignoring unsupported options: build

Creating network assignment4_default
Creating service assignment4_web-app
root@appsec-VirtualBox:/media/sf_NYU/2019_Fall_Application_Security/Assignment4#
```

Once the command has been run we can docker ps to see our running containers and we see 3 replicas which match our docker-compose.yml file.

```
root@appsec-VirtualBox:/media/sf_NYU/2019_Fall_Application_Security/Assignment4# docker ps -a
CONTAINER ID        IMAGE                 COMMAND                  CREATED             STATUS              PORTS               NAMES
d85539280ef1        assignment4:latest    "/bin/sh -c 'flask r…"   20 seconds ago      Up 13 seconds                           assignment4_web-app.2.wfjgt54h2o9j66p6larwsj1l9
f579539f1080        assignment4:latest    "/bin/sh -c 'flask r…"   20 seconds ago      Up 13 seconds                           assignment4_web-app.1.rk7bcmlcnlxsmhpm0212i2grc
c9c112d1b9b2        assignment4:latest    "/bin/sh -c 'flask r…"   22 seconds ago      Up 11 seconds                           assignment4_web-app.3.tc36b3j8o5oxhobmx9l68omk3
root@appsec-VirtualBox:/media/sf_NYU/2019_Fall_Application_Security/Assignment4# docker ps
CONTAINER ID        IMAGE                 COMMAND                  CREATED             STATUS              PORTS               NAMES
d85539280ef1        assignment4:latest    "/bin/sh -c 'flask r…"   25 seconds ago      Up 18 seconds                           assignment4_web-app.2.wfjgt54h2o9j66p6larwsj1l9
f579539f1080        assignment4:latest    "/bin/sh -c 'flask r…"   25 seconds ago      Up 17 seconds                           assignment4_web-app.1.rk7bcmlcnlxsmhpm0212i2grc
c9c112d1b9b2        assignment4:latest    "/bin/sh -c 'flask r…"   27 seconds ago      Up 16 seconds                           assignment4_web-app.3.tc36b3j8o5oxhobmx9l68omk3
root@appsec-VirtualBox:/media/sf_NYU/2019_Fall_Application_Security/Assignment4#
```

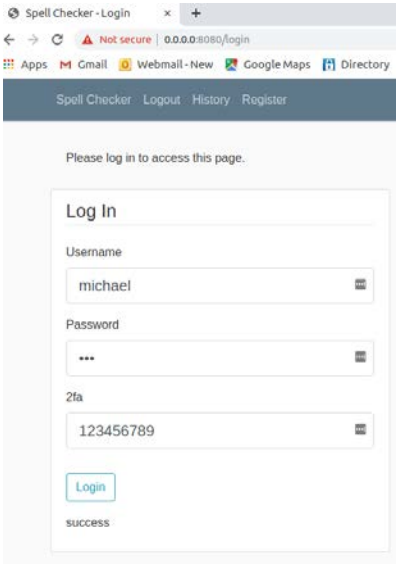
We can manually check that a particular container has the apassword and secretkey secrets by running the cat command below:

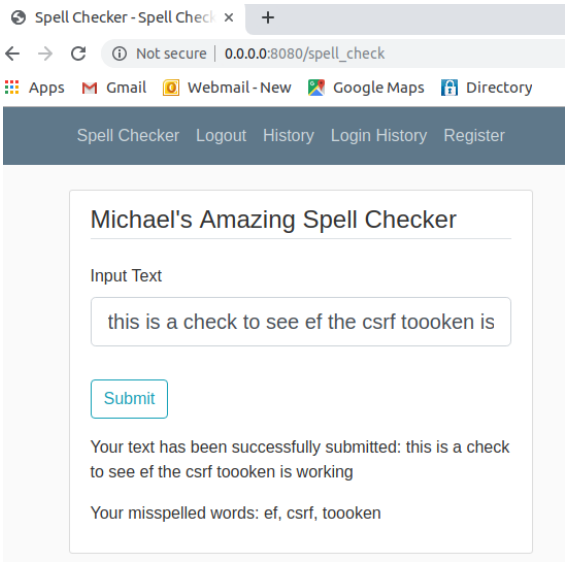
```
root@appsec-VirtualBox:/media/sf_NYU/2019_Fall_Application_Security/Assignment4# docker exec d85539280ef1 cat /run/secrets/apassword
adminpassword
root@appsec-VirtualBox:/media/sf_NYU/2019_Fall_Application_Security/Assignment4# docker exec d85539280ef1 cat /run/secrets/secretkey
thisisthesupersecretkey
root@appsec-VirtualBox:/media/sf_NYU/2019_Fall_Application_Security/Assignment4#
```

Finally we run docker stats to check that our CPU and memory is indeed limited based on the spec in our docker-compose.yml file.

```
CONTAINER ID        NAME                                                CPU %               MEM USAGE / LIMIT   MEM %               NET I/O             BLOCK I/O           PIDS
d8539280ef1         assignment4_web-app.2.wfjqt54h2o9j66p6larwsj1l9     0.02%               34.87MiB / 50MiB    69.74%              9.18kB / 0B         36.9kB / 274kB      2
f579539f1080        assignment4_web-app.1.rk7bcmlcnlxsmhpm0212i2grc     0.01%               34.79MiB / 50MiB    69.57%              9.18kB / 0B         0B / 274kB          2
c9c112d1b9b2        assignment4_web-app.3.tc36b3j8o5oxhobmx9l68omk3     0.01%               34.82MiB / 50MiB    69.65%              9.18kB / 0B         0B / 274kB          2
```

One last check with the app on the browser itself shows that logins and spellcheck posts work to ensure that our CSRF token is being used properly with a secretkey in docker secrets:





DOCKER CONTENT TRUST

One of the benefits of docker is the simplicity of building a containerized solution by simply creating a dockerfile. The dockerfile provides instructions for the build but the actual images for the pre-requisites (i.e. Ubuntu) are pulled from public or private registries. In the case of Ubuntu, we pulled this image from a public registry into our local registry and our web app (app.py) we pulled locally. If we wanted others to use app.py we would have to push to a public registry. How can one be certain that they are getting the image that I created and not some malicious version? This is issue that docker content trust addresses and does so by signing the images with trust keys (key sets).