Kevin Zhang
Unit 4 Assignment – Write-up
Link to github: https://github.com/itwonton/app_sec_home_4

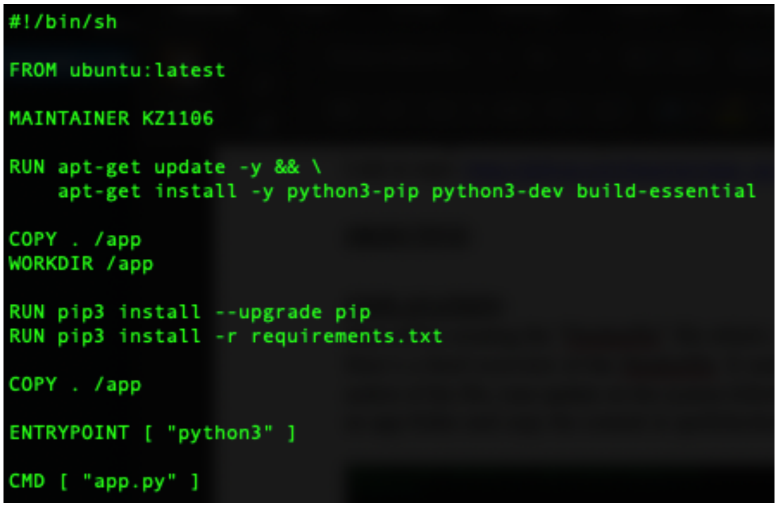
OBJECTIVE:

This assignment will we learn how to containerized our web application that we developed using Docker and other Dockers' features like Docker stack, Docker swarm, and Docker secrets.

EXPLANATION:

Dockerfile

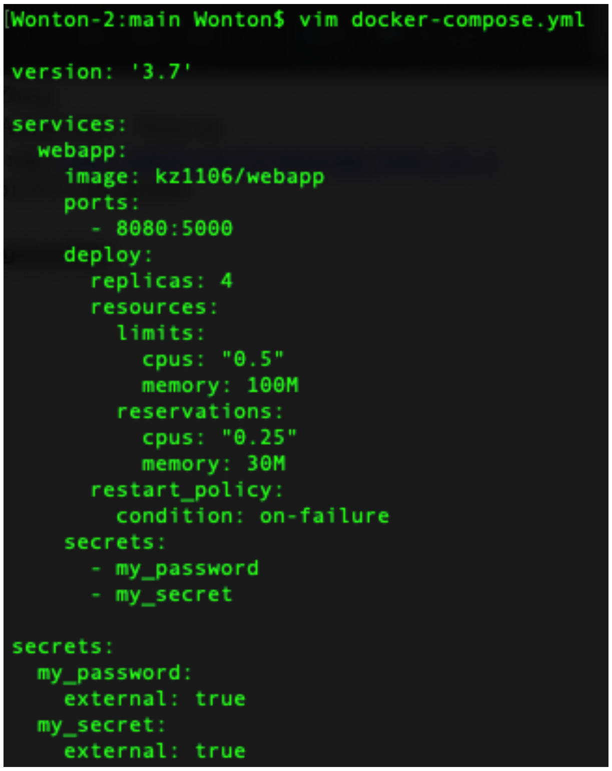
Start off by creating the "Dockerfile" file which installs the dependencies for my web application. Here's a brief overview of the Dockerfile. It uses the latest version of ubuntu, set myself as the author of the file, runs update on the system follow by installing python3 and pip3. Next, it creates an app folder and copy the content over to the app/ directory, runs pip3 update and packages from the requirements.txt file. Lastly, uses python3 to launch the app.py.



```
#!/bin/sh

FROM ubuntu:latest

MAINTAINER KZ1106

RUN apt-get update -y && \
    apt-get install -y python3-pip python3-dev build-essential

COPY . /app
WORKDIR /app

RUN pip3 install --upgrade pip
RUN pip3 install -r requirements.txt

COPY . /app

ENTRYPOINT [ "python3" ]

CMD [ "app.py" ]
```

yml file

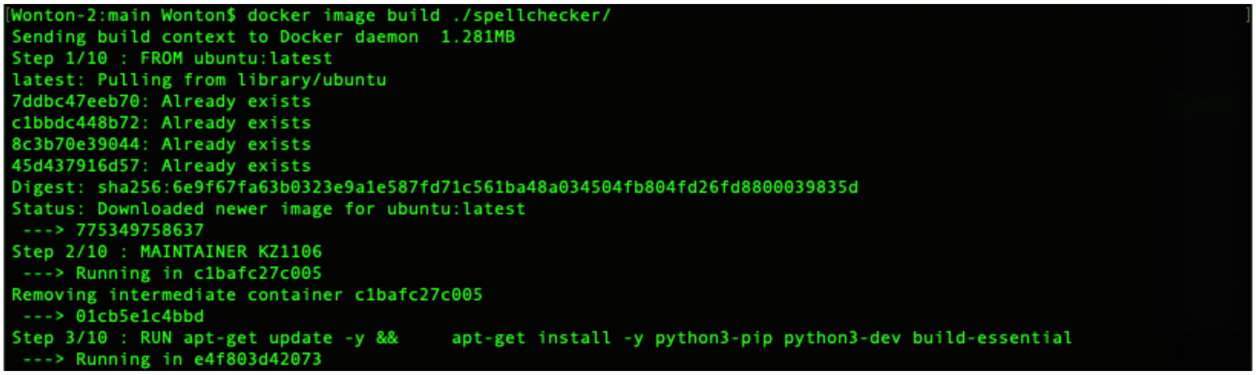
After the Dockerfile, we create the "docker-compose.yml" file which is a config file for docker-compose. It allows us to deploy, combine, and configure multiple docker container at the same time. Here's a rundown for the docker-compose.yml file. We run our image from my Docker repository which I built it out locally then pushed it to my repo. We mapped to the 5000 port to 8080. Then deploy 4 replicas with the following resources, CPU: 0.5 and memory: 100M each. Finally we set our passwords which you have to set before starting the service.



```
[Wonton-2:main Wonton$ vim docker-compose.yml

version: '3.7'

services:
  webapp:
    image: kz1106/webapp
    ports:
      - 8080:5000
    deploy:
      replicas: 4
      resources:
        limits:
          cpus: "0.5"
          memory: 100M
        reservations:
          cpus: "0.25"
          memory: 30M
      restart_policy:
        condition: on-failure
    secrets:
      - my_password
      - my_secret

secrets:
  my_password:
    external: true
  my_secret:
    external: true
```

Docker image

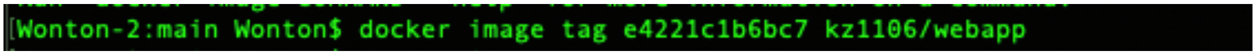
Building out the image from a Dockerfile



```
[Wonton-2:main Wonton$ docker image build ./spellchecker/                                                    ]
Sending build context to Docker daemon  1.281MB
Step 1/10 : FROM ubuntu:latest
latest: Pulling from library/ubuntu
7ddbc47eeb70: Already exists
c1bbdc448b72: Already exists
8c3b70e39044: Already exists
45d437916d57: Already exists
Digest: sha256:6e9f67fa63b0323e9a1e587fd71c561ba48a034504fb804fd26fd8800039835d
Status: Downloaded newer image for ubuntu:latest
 ---> 775349758637
Step 2/10 : MAINTAINER KZ1106
 ---> Running in c1bafc27c005
Removing intermediate container c1bafc27c005
 ---> 01cb5e1c4bbd
Step 3/10 : RUN apt-get update -y &&     apt-get install -y python3-pip python3-dev build-essential
 ---> Running in e4f803d42073
```

Docker image (continued)

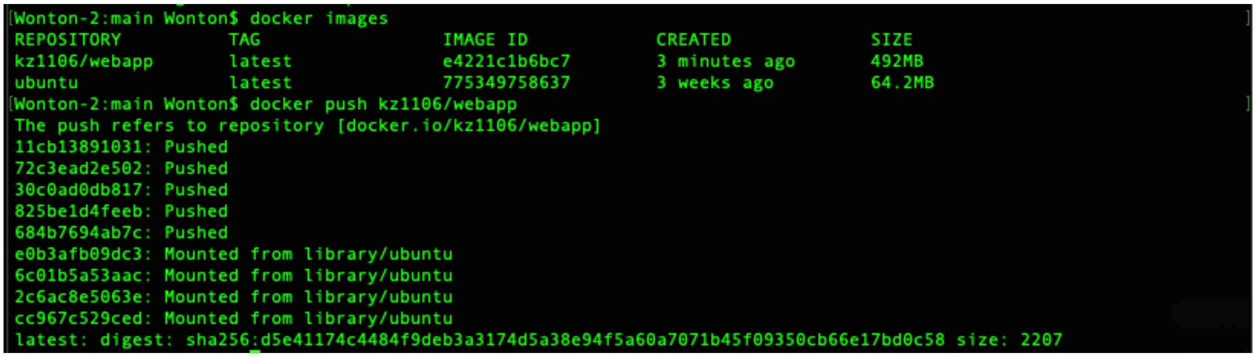
Give the image file a name before pushing it to our Docker repository



```
[Wonton-2:main Wonton$ docker image tag e4221c1b6bc7 kz1106/webapp
```

Docker image (continued)

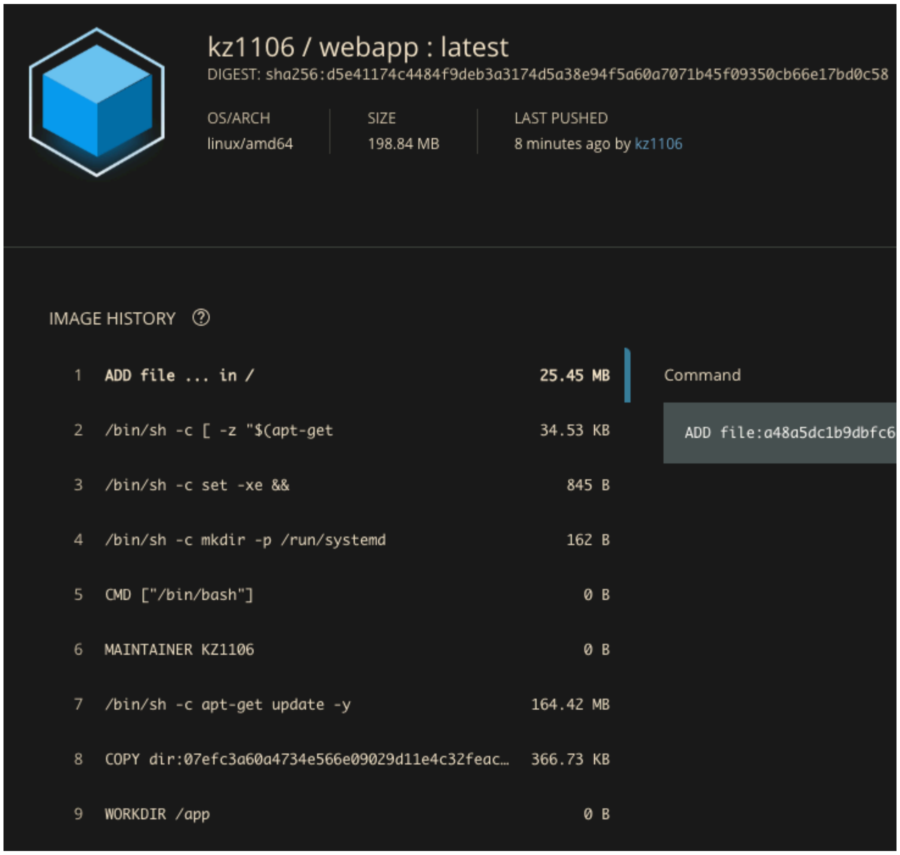
Here's a list of the images showing the image name, id, date creation, size, and etc…



```
[Wonton-2:main Wonton$ docker images                                                                    ]
 REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
 kz1106/webapp       latest              e4221c1b6bc7        3 minutes ago       492MB
 ubuntu              latest              775349758637        3 weeks ago         64.2MB
[Wonton-2:main Wonton$ docker push kz1106/webapp                                                        ]
 The push refers to repository [docker.io/kz1106/webapp]
 11cb13891031: Pushed
 72c3ead2e502: Pushed
 30c0ad0db817: Pushed
 825be1d4feeb: Pushed
 684b7694ab7c: Pushed
 e0b3afb09dc3: Mounted from library/ubuntu
 6c01b5a53aac: Mounted from library/ubuntu
 2c6ac8e5063e: Mounted from library/ubuntu
 cc967c529ced: Mounted from library/ubuntu
 latest: digest: sha256:d5e41174c4484f9deb3a3174d5a38e94f5a60a7071b45f09350cb66e17bd0c58 size: 2207
```

Docker image (continued)

Here's an image of the docker image file in our repository. When you call the image file, it runs through the Dockerfile file.



kz1106 / webapp : latest

DIGEST: sha256:d5e41174c4484f9deb3a3174d5a38e94f5a60a7071b45f09350cb66e17bd0c58

OS/ARCH linux/amd64

SIZE 198.84 MB

LAST PUSHED 8 minutes ago by kz1106

IMAGE HISTORY

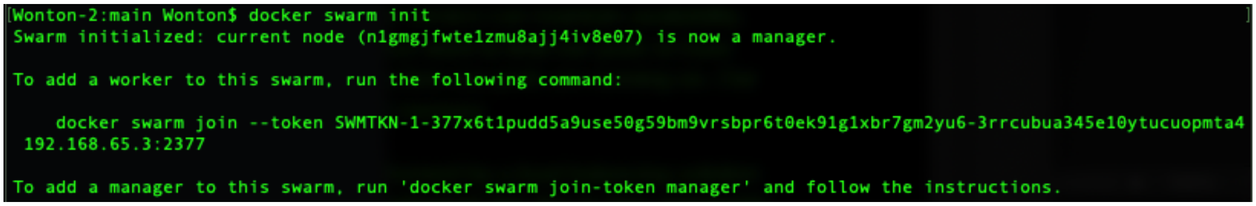
1	ADD file ... in /	25.45 MB
2	/bin/sh -c [-z "$(apt-get	34.53 KB
3	/bin/sh -c set -xe &&	845 B
4	/bin/sh -c mkdir -p /run/systemd	162 B
5	CMD ["/bin/bash"]	0 B
6	MAINTAINER KZ1106	0 B
7	/bin/sh -c apt-get update -y	164.42 MB
8	COPY dir:07efc3a60a4734e566e09029d11e4c32feac…	366.73 KB
9	WORKDIR /app	0 B

Command

ADD file:a48a5dc1b9dbfc6

Docker Swarm

Kick off swarm by running ‘Docker swarm init’ which creates a single-node swarm on the current node (our local host). We can add other virtualbox, or hyper-v hosts to join our swarm by running the command displayed in the image below, ‘docker swarm join –token…’.

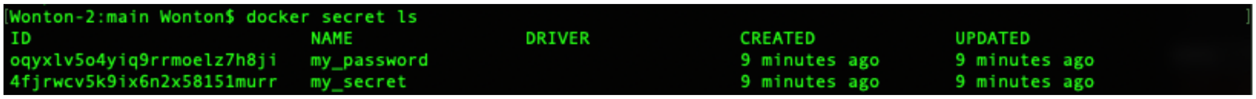


```
[Wonton-2:main Wonton$ docker swarm init
 Swarm initialized: current node (n1gmgjfwte1zmu8ajj4iv8e07) is now a manager.

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

     docker swarm join --token SWMTKN-1-377x6t1pudd5a9use50g59bm9vrsbpr6t0ek91g1xbr7gm2yu6-3rrcubua345e10ytucuopmta4
  192.168.65.3:2377

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

Docker Swarm (continued)

The following image shows that the password has been created for my_password & my_secret. In order to create these passwords we would run the following commands:

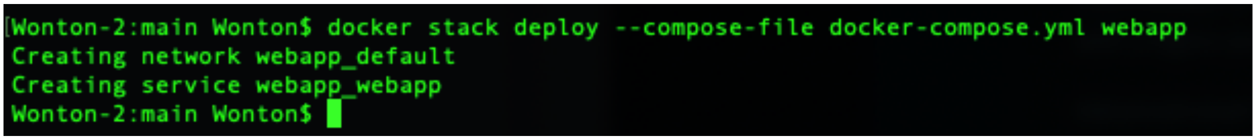
- echo ‘<password>’ | docker secret create my_password –
- echo ‘<password>’ | docker secret create my_secret –



```
[Wonton-2:main Wonton$ docker secret ls
 ID                          NAME                DRIVER              CREATED             UPDATED
 oqyxlv5o4yiq9rrmoelz7h8ji   my_password                             9 minutes ago       9 minutes ago
 4fjrwcv5k9ix6n2x58151murr   my_secret                               9 minutes ago       9 minutes ago
```

Docker Swarm (continued)

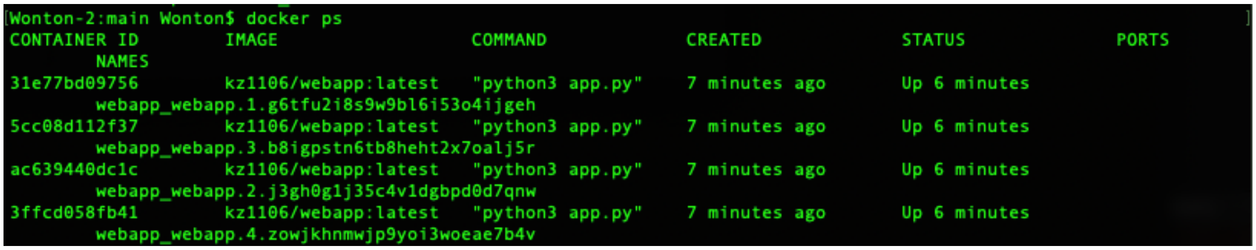
We now run the ‘docker stack deploy –compose-file docker-compose.yml webapp’ which creates a new stack or update an existing stack. The first parameter looks into the yml file with all the configuration and follow by ‘webapp’ the name we call our service.



```
[Wonton-2:main Wonton$ docker stack deploy --compose-file docker-compose.yml webapp
 Creating network webapp_default
 Creating service webapp_webapp
 Wonton-2:main Wonton$ 
```

Docker Swarm (continued)

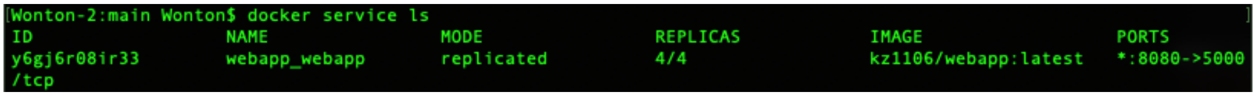
Verified that the four replicas, we specified in our yml, are up by running the command ‘docker ps’ which shows only running containers.



```
[Wonton-2:main Wonton$ docker ps
 CONTAINER ID        IMAGE                  COMMAND              CREATED             STATUS              PORTS
         NAMES
 31e77bd09756        kz1106/webapp:latest   "python3 app.py"     7 minutes ago       Up 6 minutes
         webapp_webapp.1.g6tfu2i8s9w9bl6i53o4ijgeh
 5cc08d112f37        kz1106/webapp:latest   "python3 app.py"     7 minutes ago       Up 6 minutes
         webapp_webapp.3.b8igpstn6tb8heht2x7oalj5r
 ac639440dc1c        kz1106/webapp:latest   "python3 app.py"     7 minutes ago       Up 6 minutes
         webapp_webapp.2.j3gh0g1j35c4v1dgbpd0d7qnw
 3ffcd058fb41        kz1106/webapp:latest   "python3 app.py"     7 minutes ago       Up 6 minutes
         webapp_webapp.4.zowjkhnmwjp9yoi3woeae7b4v
```

Docker Swarm (continued)

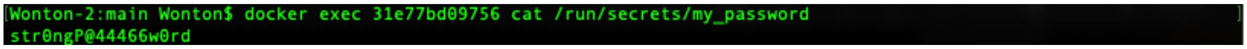
Verified that our webapp service is up and running by running ‘docker service ls’ which list the services that are running in swarm.



```
[Wonton-2:main Wonton$ docker service ls                                                                    ]
 ID                 NAME               MODE               REPLICAS           IMAGE                 PORTS
 y6gj6r08ir33       webapp_webapp      replicated         4/4                kz1106/webapp:latest  *:8080->5000
 /tcp
```

Docker Swarm (continued)

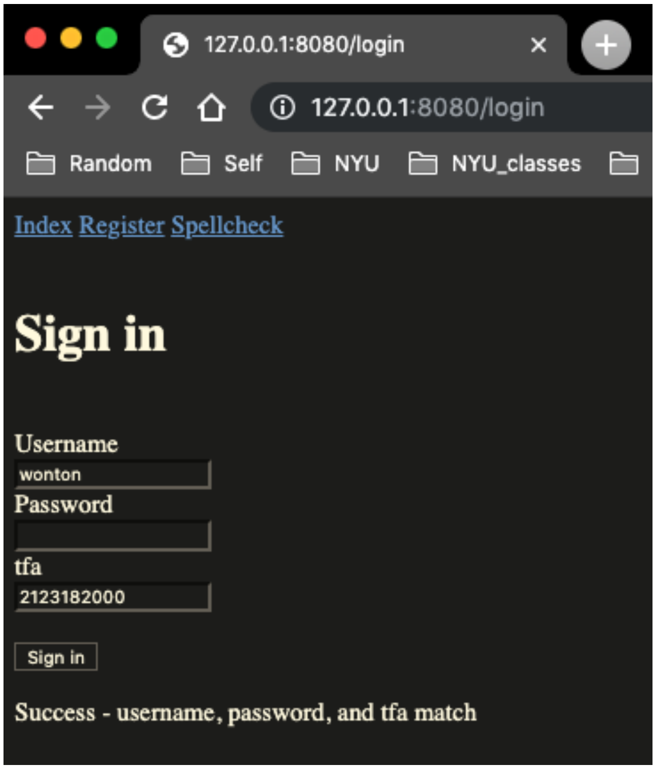
We were successfully able to display our webapp password by running the following command, ‘docker exec <container id> cat /run/secrets/my_password’. The ‘docker exec’ command runs a new command in a running container.



```
[Wonton-2:main Wonton$ docker exec 31e77bd09756 cat /run/secrets/my_password                                ]
 str0ngP@44466w0rd
```

Webapp login

To test the service we navigated to our localhost and specifying the 8080 port. Next, we were able to login successfully with our test account as you can see below.



Sign in

Username
wonton
Password

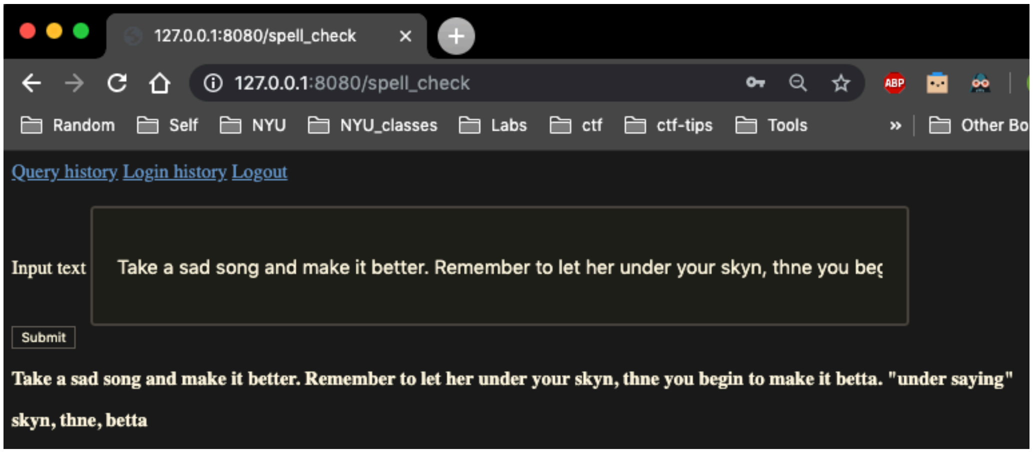
tfa
2123182000

Sign in

Success - username, password, and tfa match

Webapp spell check

To test the spell check function we navigated to our spell_check page. Next, we entered a couple of sentences with some incorrect words. After hitting submit, it was able to spit back the words that were incorrect.



Input text: Take a sad song and make it better. Remember to let her under your skyn, thne you beg

Take a sad song and make it better. Remember to let her under your skyn, thne you begin to make it betta. "under saying"

skyn, thne, betta

Docker Content Trust

Docker content trust adds the extra layer of security. It gives the team ability to sign images and protect against man in the middle attacks. It also gives them the ability to see who has created images and ensure that teams are running the latest version.

Basically the Docker engine signs the image locally with the publisher's key and then pushes the image to the registry. Anyone who wants to use your image pulls the image down from the registry, and then the user uses Docker engine to verify the image is not tampered with and up to date.