نام و نام خانوادگی: سروش مهربان
شماره دانشجویی: 9628099

تمرین دوم رایانش ابری

گام اول:

برای این گام یک Dockerfile درست شده است که از alpine استفاده شده و روی آن curl را نصب میکنیم و image آن را با دستور زیر تشکیل می‌دهیم:

```
docker build -t new-alpine:1.0 .
```

حال می‌خواهیم این image ساخته شده را در Docker Hub قرار دهیم. برای این کار ابتدا یک تگ جدید اضافه می‌کنیم:

```
docker tag new-alpine:1.0 soroushmehraban2022/new-alpine:1.0
```

سپس با دستور زیر image ساخته شده را در Docker Hub پوش می‌کنیم:

```
docker image push soroushmehraban2022/new-alpine:1.0
```

اسکرین شات زیر، پس از عمل push را نشان می‌دهد:

```
C:\Users\Asus>docker image push soroushmehraban2022/new-alpine:1.0
The push refers to repository [docker.io/soroushmehraban2022/new-alpine]
89e8a9e42b4b: Pushed
a1c01e366b99: Mounted from library/alpine
1.0: digest: sha256:a73d8a6c601ad843a816be13c58c460b7499fba199b0b3260c34c9ec3feb0a21 size: 739
```

حال برای تست دریافت image از Docker Hub، ابتدا image ساخته شده را از داخل کامپیوتر با دستور زیر حذف می‌کنیم:

```
C:\Users\Asus>docker rmi soroushmehraban2022/new-alpine:1.0
Untagged: soroushmehraban2022/new-alpine:1.0
Untagged: soroushmehraban2022/new-alpine@sha256:a73d8a6c601ad843a816be13c58c460b7499fba199b0b3260c34c9ec3feb0a21

C:\Users\Asus>docker images
REPOSITORY                  TAG           IMAGE ID       CREATED          SIZE
new-alpine                  1.0           41eb3e1fdbdc   18 minutes ago   7.66MB
keycloak                    1.0           166807c2efd9   3 weeks ago      719MB
rabbitmq                    3-management  c8817e468079   3 weeks ago      254MB
postgres                    latest        f98d2cc7f971   4 weeks ago      374MB
quay.io/keycloak/keycloak   17.0.0        81417e4be7ec   6 weeks ago      556MB
gcr.io/k8s-minikube/kicbase v0.0.30       1312ccd2422d   7 weeks ago      1.14GB
recipe-app-api_app          latest        980de5ab5ea7   7 weeks ago      556MB
ubuntu                      latest        d13c942271d6   2 months ago     72.8MB
postgres                    10-alpine     eeb19929169c   2 months ago     76.3MB
wurstmeister/kafka          latest        2dd91ce2efe1   2 months ago     508MB
wurstmeister/zookeeper      latest        3f43f72cb283   3 years ago      510MB

C:\Users\Asus>docker rmi new-alpine:1.0
Untagged: new-alpine:1.0
Deleted: sha256:41eb3e1fdbdce8d88947fb2eb88b253308abea1b1e1c680e03caf398ebeb06da

C:\Users\Asus>docker images
REPOSITORY                  TAG           IMAGE ID       CREATED          SIZE
keycloak                    1.0           166807c2efd9   3 weeks ago      719MB
rabbitmq                    3-management  c8817e468079   3 weeks ago      254MB
postgres                    latest        f98d2cc7f971   4 weeks ago      374MB
quay.io/keycloak/keycloak   17.0.0        81417e4be7ec   6 weeks ago      556MB
gcr.io/k8s-minikube/kicbase v0.0.30       1312ccd2422d   7 weeks ago      1.14GB
recipe-app-api_app          latest        980de5ab5ea7   7 weeks ago      556MB
ubuntu                      latest        d13c942271d6   2 months ago     72.8MB
postgres                    10-alpine     eeb19929169c   2 months ago     76.3MB
wurstmeister/kafka          latest        2dd91ce2efe1   2 months ago     508MB
wurstmeister/zookeeper      latest        3f43f72cb283   3 years ago      510MB
```

با توجه به تصویر فوق مشاهده میکنیم که هر ۲ image با موفقیت حذف شده‌اند.

سپس با دستور زیر image را از Docker Hub دریافت می‌کنیم:

```
C:\Users\Asus>docker image pull soroushmehraban2022/new-alpine:1.0
1.0: Pulling from soroushmehraban2022/new-alpine
40e059520d19: Already exists
ad73f0286d2a: Already exists
Digest: sha256:a73d8a6c601ad843a816be13c58c460b7499fba199b0b3260c34c9ec3feb0a21
Status: Downloaded newer image for soroushmehraban2022/new-alpine:1.0
docker.io/soroushmehraban2022/new-alpine:1.0

C:\Users\Asus>docker images
REPOSITORY                      TAG           IMAGE ID       CREATED          SIZE
soroushmehraban2022/new-alpine  1.0           41eb3e1fdbdc   20 minutes ago   7.66MB
keycloak                        1.0           166807c2efd9   3 weeks ago      719MB
rabbitmq                        3-management  c8817e468079   3 weeks ago      254MB
postgres                        latest        f98d2cc7f971   4 weeks ago      374MB
quay.io/keycloak/keycloak       17.0.0        81417e4be7ec   6 weeks ago      556MB
gcr.io/k8s-minikube/kicbase     v0.0.30       1312ccd2422d   7 weeks ago      1.14GB
recipe-app-api_app              latest        980de5ab5ea7   7 weeks ago      556MB
ubuntu                          latest        d13c942271d6   2 months ago     72.8MB
postgres                        10-alpine     eeb19929169c   2 months ago     76.3MB
wurstmeister/kafka              latest        2dd91ce2efe1   2 months ago     508MB
wurstmeister/zookeeper          latest        3f43f72cb283   3 years ago      510MB
```

با توجه به اسکرین شات فوق مشاهده می‌کنیم که با موفقیت از Docker Hub دریافت شده است.

حال image دانلود شده را اجرا میکنیم تا یک container از آن ساخته شود و curl را اجرا می‌کنیم:

```
C:\Users\Asus>docker run -it --rm soroushmehraban2022/new-alpine:1.0
/ # curl google.com
<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
<A HREF="http://www.google.com/">here</A>.
</BODY></HTML>
/ #
```

گام دوم:

برای ساخت سرور از FastAPI پایتون استفاده شده است. به طوری که آدرس API گرفتن آب و هوا را از طریق Environment variable دریافت می‌کند و پس از رکوست زدن به کمک requests پایتون، نتیجه را به همراه hostname به کاربر برمی‌گرداند.

برای build کردن:

```
E:\AUT\Cloud Computing\HW\HW2\2. WeatherServer>docker build -t weather-server:1.0 .
[+] Building 11.9s (11/11) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 32B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/python:3.7-alpine
 => [1/6] FROM docker.io/library/python:3.7-alpine@sha256:43ab3e374d4a16ac3132ad60fa150837003a093fae81f1b46cc4da8bd69129ba
 => [internal] load build context
 => => transferring context: 64B
 => CACHED [2/6] WORKDIR /app
 => CACHED [3/6] COPY requirements.txt ./
 => CACHED [4/6] RUN pip install -r requirements.txt
 => CACHED [5/6] COPY main.py ./
 => CACHED [6/6] RUN weather_url=$weather_url
 => exporting to image
 => => exporting layers
 => => writing image sha256:59eee55b094798e73d4370aabe9e5f2d3722dca93e957e06e16f91692faf07ee
 => => naming to docker.io/library/weather-server:1.0

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
```

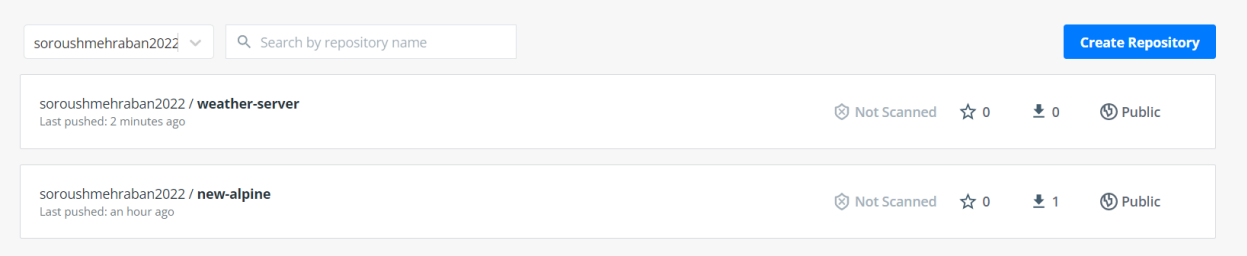
(با توجه به این که بار اول دستم خورد زدم پاک کردم مشاهده میکنیم بار دوم همرو از cache برداشته است).

سپس در Docker Hub قرار می‌دهیم:

```
E:\AUT\Cloud Computing\HW\HW2\2. WeatherServer>docker tag weather-server:1.0 soroushmehraban2022/weather-server:1.0

E:\AUT\Cloud Computing\HW\HW2\2. WeatherServer>docker image push soroushmehraban2022/weather-server:1.0
The push refers to repository [docker.io/soroushmehraban2022/weather-server]
4ee2493d7111: Pushed
23257d70fa37: Pushed
cd24679f828f: Pushed
5fb2a3710871: Pushed
835b2b5370d2: Pushed
4d267d08398c: Pushed
c683d6c2cad6: Mounted from library/python
ec4c4cc34642: Mounted from library/python
c2b3a6e662b5: Mounted from library/python
a1c01e366b99: Mounted from library/python
1.0: digest: sha256:015732c95ccbfec3fd37149ed0bf2f625e3723d6917d01e31284565ceeafa367 size: 2407
```

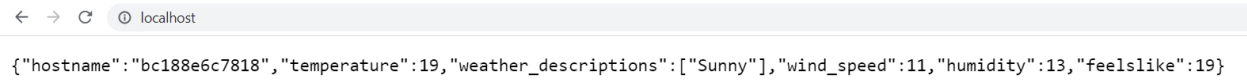
مشاهده می‌کنیم image موردنظر ما در Docker Hub قرار گرفته است:



برای تست در سیستم شخصی نیز با دستور زیر آن را اجرا می‌کنیم:

```
E:\AUT\Cloud Computing\HW\HW2>docker run --rm --env weather_url="http://api.weatherstack.com/current?access_key=b3324a714ee5cdb9ec8d2bf1d83b824c&query=Tehran" -p 80:8080 soroushmehraba
n2022/weather-server:1.0
INFO:     Started server process [1]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
INFO:     172.17.0.1:44482 - "GET / HTTP/1.1" 200 OK
```

با توجه به این دستور، weather_url تحت عنوان environment variable به کانتینر داده شده است. همچنین پورت ۸۰۸۰ به صورت دیفالت برای سرور در صورت نبودenvironment variable ست می‌شود که آن نیز به پورت ۸۰ دستگاه نگاشته شده است. پس با مشاهده در browser داریم:



گام سوم:

در این گام ابتدا ConfigMap مورد نیاز را در یک فایل yaml ایجاد می‌کنیم:

server-config.yaml

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: server-config
data:
  weather_url: "http://api.weatherstack.com/current?access_key=b3324a714ee5cdb9ec8d2bf1d83b824c&query=Tehran"
  server_port: "8000"

```

سپس با دستور زیر configmap را ساخته و در لیست configmap ها با دستور بعدی مشاهده می‌کنیم:

```
E:\AUT\Cloud Computing\HW\HW2\3. Kubernetes>kubectl apply -f server-config.yaml
configmap/server-config created

E:\AUT\Cloud Computing\HW\HW2\3. Kubernetes>kubectl get configmaps
NAME               DATA   AGE
kube-root-ca.crt   1      9h
server-config      2      12s
```

در ادامه یک deployment با دستور زیر میسازیم:

kubectl create deployment server-deployment --image=soroushmehraban2022/weather-server:1.0 --run-dry=client -o yaml > server-deployment.yaml

و در env، مقادیر را از configmap می‌خوانیم. سپس آن را نیز اپلای میکنیم و با دستور get مشاهده می‌کنیم که با موفقیت ایجاد شده است:

```
E:\AUT\Cloud Computing\HW\HW2\3. Kubernetes>kubectl get deployments
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
server-deployment   2/2     2            2           148m
```

سپس یک سرویس ایجاد می‌کنیم که deployment را روی پورت ۸۰ اجرا کند. سپس آن را نیز get میکنیم و داریم:

```
E:\AUT\Cloud Computing\HW\HW2\3. Kubernetes>kubectl get service
NAME             TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
kubernetes       ClusterIP   10.96.0.1      <none>        443/TCP   11h
server-service   ClusterIP   10.97.53.201   <none>        80/TCP    4m36s
```

علت استفاده از سرویس: همانطور که در workshop توضیح داده شد، پادها پس از ایجاد هربار یک IP به آن ها اختصاص می‌یابد و در صورت از بین رفتن deployment به صورت خودکار پاد جدیدی را ایجاد می‌کند. اما لزوما همان IP قبلی به آن اختصاص نمی‌یابد. پس یک سرویس ایجاد میکنیم که به طور خودکار با این پادها در ارتباط باشد و ترافیک را بین آن ها توزیع کند.

پاد های در حال حاضر:

```
E:\AUT\Cloud Computing\HW\HW2\3. Kubernetes>kubectl get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE    IP           NODE       NOMINATED NODE   READINESS GATES
server-deployment-5b8d668949-gf8wv   1/1     Running   0          164m   172.17.0.3   minikube   <none>           <none>
server-deployment-5b8d668949-rnktd   1/1     Running   0          164m   172.17.0.4   minikube   <none>           <none>
```

که همانطور که مشاده میکنیم، هردو پاد برای server-deployment می‌باشند و جلوی آن یک تگ گذاشته شده است که مشخص شود اگر بعدا تغییری در deployment داشتیم این پادها برای تغییرات جدید هستن یا قدیم و جلوی آن نیز یک تگ دیگر گذاشته شده است که تمایز بین پادها را نشان دهد.

برای ارتباط بین پادها و سرویس ساخته شده بایستی به endpoints توجه کنیم:

```
E:\AUT\Cloud Computing\HW\HW2\3. Kubernetes>kubectl get ep
NAME             ENDPOINTS                         AGE
kubernetes       192.168.49.2:8443                 12h
server-service   172.17.0.3:8000,172.17.0.4:8000   22m
```

همانطور که مشاهده می‌کنیم، server-service با ۲ تا پاد مورد نظر ارتباط برقرار گرفته است و IP آن ۲ را در بخش ENDPOINTS نوشته است.

گام چهارم:

ابتدا با دستور kubectl run سعی می‌کنیم image ساخته شده در گام ۱ را از آن یک pod درست کنیم:

```
E:\AUT\Cloud Computing\HW\HW2\4. Testing the system>kubectl run curl-pod --image=soroushmehraban2022/new-alpine:1.0
pod/curl-pod created

E:\AUT\Cloud Computing\HW\HW2\4. Testing the system>kubectl get pods
NAME                                READY   STATUS             RESTARTS      AGE
curl-pod                            0/1     CrashLoopBackOff   1 (6s ago)    8s
server-deployment-5b8d668949-gf8wv  1/1     Running            1 (23m ago)   15h
server-deployment-5b8d668949-rnktd  1/1     Running            1 (23m ago)   15h
```

مشاهده می‌کنیم status آن به CrashLoopBackOff رسیده است.

علت: با ساختن container چون کاری برای انجام دادن ندارد، به سرعت بسته می‌شود و بعد دوباره تلاش به ساختن آن می‌کند و بسته می‌شود و این حلقه مدام تکرار می‌شود تا زمانی که دیگر Crash بکند و به پایان برسد.

راه حل: یک deployment میسازیم بطوری که هنگام اجرای container دستور /bin/sleep را به مدت بی‌نهایت اجرا کند. با این کار دیگر کارش به پایان نمی‌رسد و میتوانیم دسترسی به ash را بگیریم.

پس در ابتدا pod ساخته شده را حذف می‌کنیم:

```
docker delete pod curl-pod
```

سپس مشابه قبل یک deployment با دستور زیر می‌سازیم:

kubectl create deployment curl-deployment --image=soroushmehraban2022/new-alpine:1.0 --run-dry=client -o yaml > curl-deployment.yaml

در اسکرین شات زیر مشاهده می‌کنیم که deployment مورد نظر ساخته شده است و در دستور بعدی نیز مشاهده می‌کنیم پاد مورد نظر ما در وضعیت Running قرار گرفته است.

```
E:\AUT\Cloud Computing\HW\HW2\4. Testing the system>kubectl get deployments
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
curl-deployment     1/1     1            1           37s
server-deployment   2/2     2            2           15h

E:\AUT\Cloud Computing\HW\HW2\4. Testing the system>kubectl get pods
NAME                                READY   STATUS    RESTARTS      AGE
curl-deployment-6f95cf5f47-lrt6c    1/1     Running   0             43s
server-deployment-5b8d668949-gf8wv  1/1     Running   1 (11m ago)   15h
server-deployment-5b8d668949-rnktd  1/1     Running   1 (11m ago)   15h
```

```
E:\AUT\Cloud Computing\HW\HW2\4. Testing the system>kubectl exec curl-deployment-6f95cf5f47-lrt6c -it -- ash
/ # curl server-service
{"hostname":"server-deployment-5b8d668949-rnktd","temperature":20,"weather_descriptions":["Partly cloudy"],"wind_speed":11,"humidity":12,"feelslike":20}/ #
/ # curl server-service
{"hostname":"server-deployment-5b8d668949-rnktd","temperature":20,"weather_descriptions":["Partly cloudy"],"wind_speed":11,"humidity":12,"feelslike":20}/ #
/ # curl server-service
{"hostname":"server-deployment-5b8d668949-rnktd","temperature":20,"weather_descriptions":["Partly cloudy"],"wind_speed":11,"humidity":12,"feelslike":20}/ #
/ # curl server-service
{"hostname":"server-deployment-5b8d668949-rnktd","temperature":20,"weather_descriptions":["Partly cloudy"],"wind_speed":11,"humidity":12,"feelslike":20}/ #
/ # curl server-service
{"hostname":"server-deployment-5b8d668949-gf8wv","temperature":20,"weather_descriptions":["Partly cloudy"],"wind_speed":11,"humidity":12,"feelslike":20}/ #
/ # curl server-service
{"hostname":"server-deployment-5b8d668949-gf8wv","temperature":20,"weather_descriptions":["Partly cloudy"],"wind_speed":11,"humidity":12,"feelslike":20}/ #
/ # curl server-service
{"hostname":"server-deployment-5b8d668949-rnktd","temperature":20,"weather_descriptions":["Partly cloudy"],"wind_speed":11,"humidity":12,"feelslike":20}/ #
/ # curl server-service
{"hostname":"server-deployment-5b8d668949-gf8wv","temperature":20,"weather_descriptions":["Partly cloudy"],"wind_speed":11,"humidity":12,"feelslike":20}/ #
/ #
```

در آخر با توجه به دستور فوق به پاد مورد نظر وصل شده و دسترسی ash را می‌گیریم. در نهایت، چند دستور curl به server-service میزنیم. مشاهده میکنیم که تمامی دستورات با موفقیت نتیجه داده اند و در قسمت hostname نیز مشاهده می‌کنیم که ترافیک بین ۲پاد توزیع شده است.