

گزارش عملی فاز ۰ و ۱ داکر

یاسمن گودرزی ۹۹۳۱۱۰۰

در این فاز ابتدا از ما خواسته شد تا داکر فایل مربوط به دستورات را بنویسیم. داکر فایل به صورت زیر است:

```
▼ DOCKER Dockerfile > ...
  cc_hw2_hello.py
  Dockerfile
1 FROM python:3.9.19-alpine
2 COPY . /app
3 WORKDIR /app
4 RUN pip install numpy
5 CMD python3 cc_hw2_hello.py
```

با کامند زیر از داکر فایل خودمون داکر ایمج به نام hello\_docker می سازیم  
sudo docker build -t hello\_docker .

ایمیج داکر های ما قبل از ساخت:

```
yasaman@yasaman-VlvoBook-ASUSLaptop-X509JP-R521JP:~$ sudo docker image ls
REPOSITORY          TAG          IMAGE ID      CREATED       SIZE
cc_hw2_hello_docker latest       a1a250890bb9 3 hours ago  159MB
hello_docker        latest       a1a250890bb9 3 hours ago  159MB
<none>              <none>      ba178a1511da 12 hours ago 48.2MB
<none>              <none>      185427d09440 12 hours ago 129MB
<none>              <none>      16efdf3a14ee 12 hours ago 129MB
<none>              <none>      438f2bb6b75b 12 hours ago 129MB
<none>              <none>      438c1a3a2eb7 12 hours ago 129MB
docker.elastic.co/elasticsearch/elasticsearch 8.13.0      ae392eae3103 11 days ago  1.23GB
hello-world         latest       d2c94e258dcb 11 months ago 13.3kB
```

ایمیج داکر پس از ساخت:

ران کردن ایمج:

```
yasaman@yasaman-VlvoBook-ASUSLaptop-X509JP-R521JP:~$ sudo docker run hello_docker
Warm regards from the Cloud Computing TAs! Welcome to the Docker project.

We want to simply sort the elements in an array using NumPy in cc_hw2_hello.py
The array:
[[ 3  7  1]
 [10  3  2]
 [ 5  6  7]]

Sort the whole array:
[ 1  2  3  3  5  6  7  7 10]

Sort along each row:
[[ 1  3  7]
 [ 2  3 10]
 [ 5  6  7]]

Sort along each column:
[[ 3  3  1]
 [ 5  6  2]
 [10  7  7]]
```

ایمیج pull

```
yasaman@yasaman-VlvoBook-ASUSLaptop-X509JP-R521JP:~$ docker login
Authenticating with existing credentials...
Login Succeeded
yasaman@yasaman-VlvoBook-ASUSLaptop-X509JP-R521JP:~$ docker push yasgodarzi/hello_docker:v1.0
The push refers to repository [docker.io/yasgodarzi/hello_docker]
8dce08e00abe: Pushed
5f70bf18a086: Layer already exists
6a002f072fa4: Layer already exists
4857056bad11: Layer already exists
69141b6c4721: Layer already exists
0bbac9765c1f: Layer already exists
4c9c2b9681ab: Layer already exists
d4fc045c9e3a: Layer already exists
v1.0: digest: sha256:0f255bce7c8dae045dde1afd6e508df395b658604774a3c2602221b5b5e08a65 size: 1998
yasaman@yasaman-VlvoBook-ASUSLaptop-X509JP-R521JP:~$
```

General Tags Builds Collaborators Webhooks Settings

Add a short description for this repository  
The short description is used to index your content on Docker Hub and in search engines. It's visible to users in search results.

Update

yasgodarzi/hello\_docker

Updated 3 minutes ago

This repository does not have a description

Docker commands

To push a new tag to this repository:

docker push yasgodarzi/hello\_docker:tagname

Public View

Tags

This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
v1.0	linux/amd64	Image	---	3 minutes ago

See all

Automated Builds

Manually pushing images to Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating.

Available with Pro, Team and Business subscriptions. [Read more about automated builds](#)

Upgrade

yasgodarzi/hello\_docker:v1.0

Delete Tag

MANIFEST DIGEST

sha256:0f255bce7c8dae045dde1afd6e508df395b658604774a3c2602221b5b5e08a65

OS/ARCH	COMPRESSED SIZE	LAST PUSHED	TYPE	MANIFEST DIGEST
linux/amd64	70.33 MB	4 minutes ago by yasgodarzi	Image	sha256:0f255bce...

Image Layers

Vulnerabilities

IMAGE LAYERS

1

ADD file ... in /

3.25 MB

2

CMD ["/bin/sh"]

0 B

3

ENV PATH=/usr/local/bin:/usr/local/sbin:/usr/local/\_

0 B

4

ENV LANG=C.UTF-8

0 B

5

RUN /bin/sh -c set -eux;

605.00 KB

6

ENV GPQ\_KEY=E3FF2839C048B25C084DEBE9826995E310250568

0 B

7

ENV PYTHON\_VERSION=3.9.19

0 B

8

RUN /bin/sh -c set -eux;

11.07 MB

9

RUN /bin/sh -c set -eux;

241 B

Command

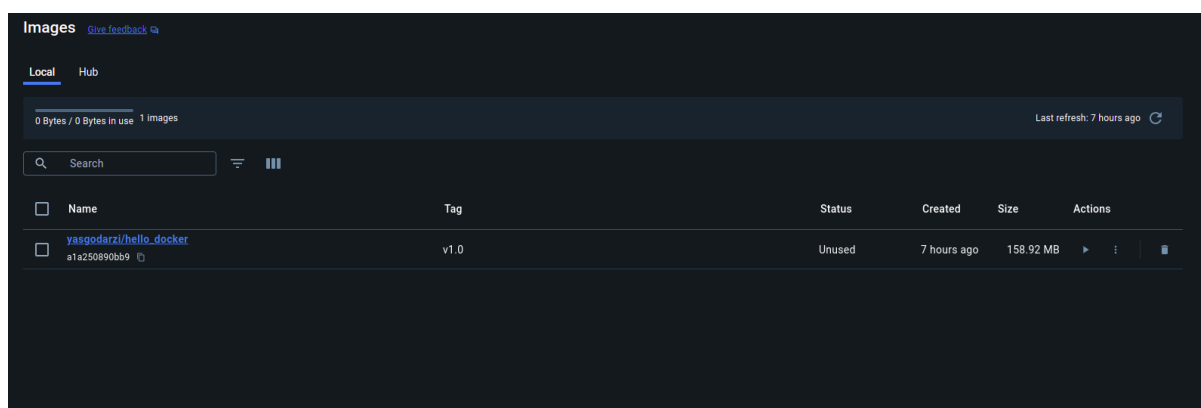
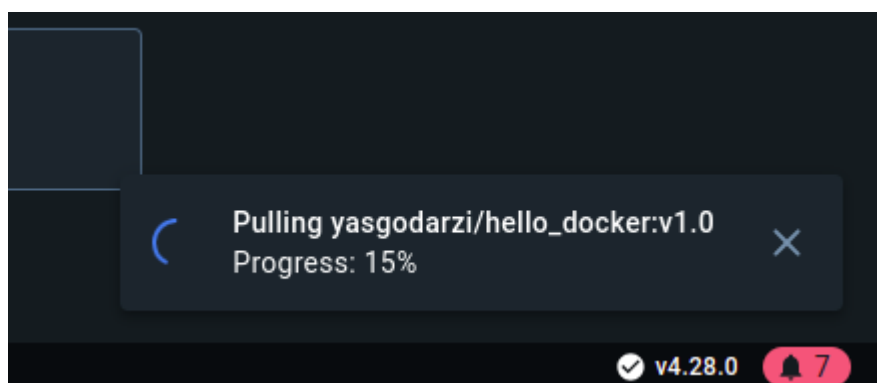
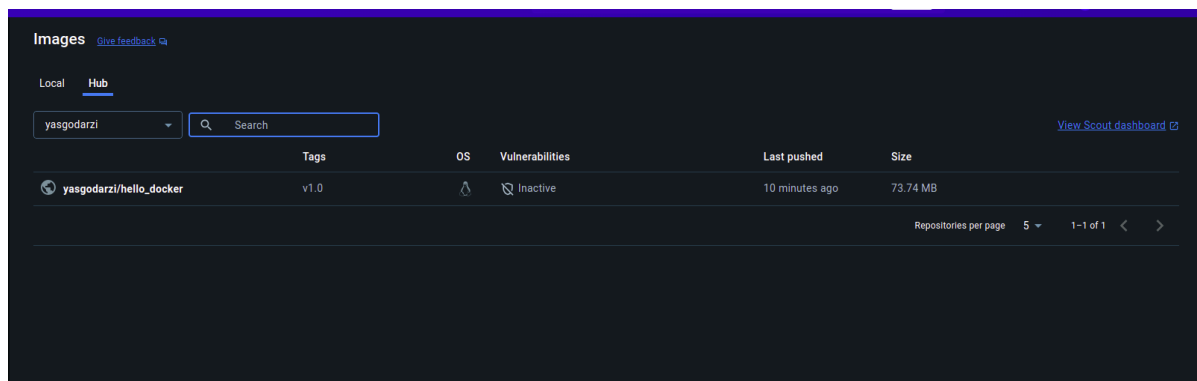
ADD file:37a76ec18f9887751cd8473744917d08b7431fc4085997bb6a09d81b41775473 in /

## ایمیج pull

با استفاده از داکر دسکتاپ ایمیج را پول میکنیم (بر روی ایمیج نگه میداریم و با ظاهر شدن دکمه pull روی ان کلیک میکنیم).

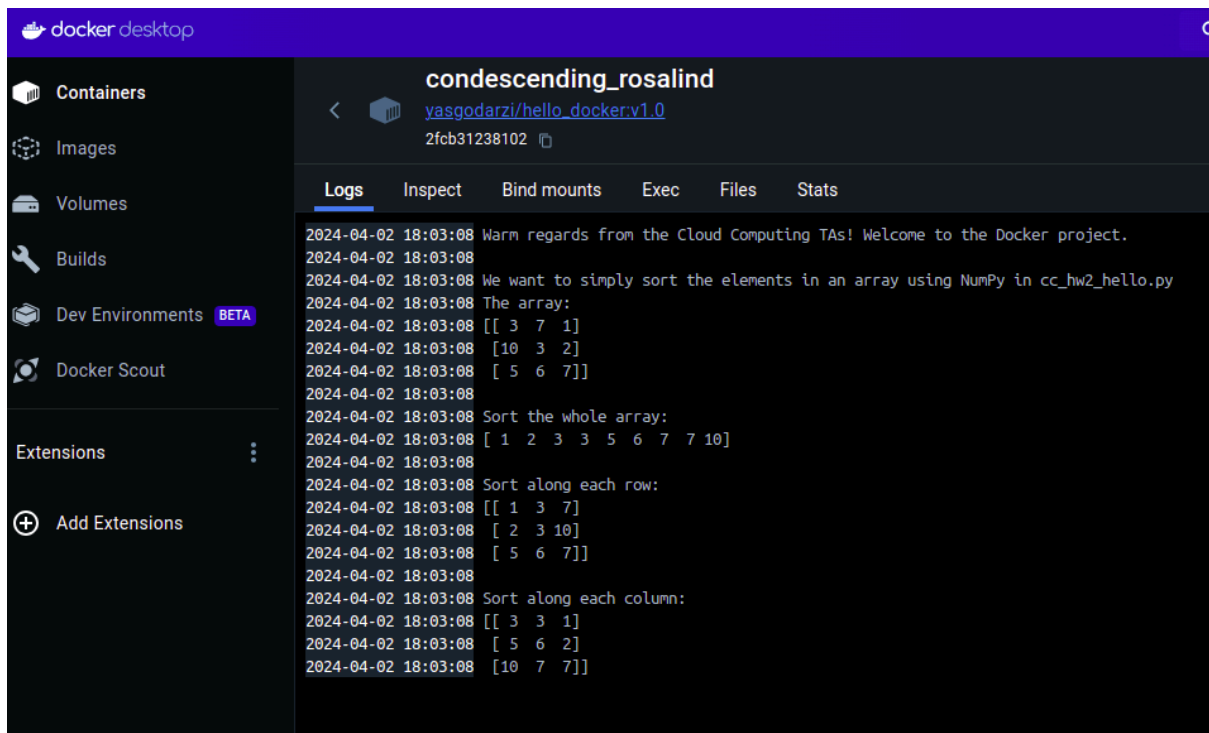
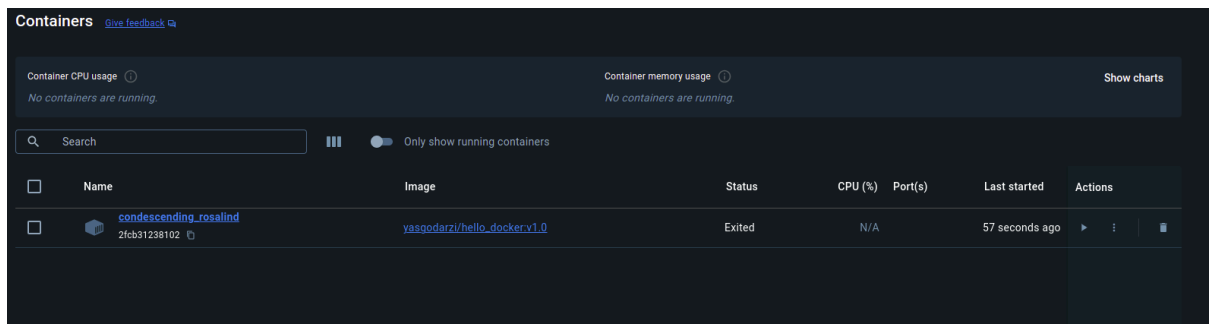
میتوان این کار را با دستور زیر هم انجام داد

```
docker pull yasgodarzi/hello_docker:v1.0
```



افزوده شدن ایمیج به ایمیج لوکال

ساخت و ران کردن کانینر



```
yasanen@yasanen-VlvoBook-ASUSLaptop-X509JP-R521JP:~$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
2fcb31238102	yasgodarzi/hello_docker:v1.0	"/bin/sh -c 'python3..."	About a minute ago	Exited (0) About a minute ago		condescending_rosalind

بخش دوم ) تفاوت run با cmd:

هر دو کامند در نوشتن داکر فایل و برای استفاده داکر ایمج و کانتینر هستند اما با این تفاوت که کامند run در هنگام پروسه ساختن ایمج از داکر فایل ما اجرا میشود. در نتیجه خروجی این دستور برروی ایمج ما اثر می گذارد. پس در واقع هرگاه خواستیم که کامندی را در هنگام تشکیل ایمج اجرا کنیم یا در ایمج تاثیر بگذاریم(مانند دانلود پکیج ها و یا کانفیگ کردن) از این کامند استفاده میکنیم. اما کامند cmd هنگامی اجرا می شود که کانتینر ما شروع به اجرا کردن بکند و تاثیری در ایمج ندارد. پس دستوراتی که می خواهیم هنگام اجرا کانتینر اجرا شوند را در این قسمت می گذاریم. به طور مثال ران شدن یک فایل.

بخش سوم)

هنگام ساختن ایمیج از داکر فایل ما، هر دستور **run** یک لایه جدید ایجاد می کند و در آن لایه اجرا میشود. در داکر فایل نوشته شده ما ابتدا با اجرای خط زیر ما ابتدا یک فایل با نام **disk.img** در لایه جدید میسازیم

```
RUN dd if=/dev/random of=/home/disk.img bs=1MB count=200
```

در خط بعدی که با دستور زیر روبرو میشویم و میخواهیم فایل ساخته شده در لایه قبل را پاک کنیم. در این لایه جدید فقط حذف فایل **disk.img** اعمال می شود. اما فایل اصلی **disk.img** که توسط دستور **dd** ایجاد شده بود، هنوز در لایه قبلی (بخشی که **dd** را اجرا می کند) وجود دارد.

```
RUN rm /home/disk.img
```

اگرچه فایل **disk.img** در دستور **rm** حذف شده است، اما اندازه فیزیکی آن هنوز در لایه قبلی وجود دارد و باعث افزایش اندازه نهایی ایمیج می شود.

راه حل: بهتر است تا دستورات را در یک خط ران در داکر فایل و به صورت زیر بنویسیم.

```
FROM busybox:latest
```

```
RUN dd if=/dev/random of=/home/disk.img bs=1MB count=200 && rm
```

```
/home/disk.im
```

(بخش چهارم)

برای اینکار میتوان ابتدا چک کرد که فایل در لایه ایمیج باقی مانده است یا خیر اگر باقی مانده بود میتوانیم آن را کپی کنیم.

با دستور زیر ایمیج را به صورت کانتینر بالا می آوریم

```
docker run -it --rm abazshoushtari/cc-ta-hw2docker-ctf:latest /bin/sh
```

سپس خروجی دستور زیر را بررسی میکنیم

```
/ls /home
```


چون خالی هست پس نشان میدهد که فایل رمز از ایمیج پاک شده است و دیگر نمی توان آن را بازیابی کرد.

```
yasaman@yasaman-VivoBook-ASUSLaptop-X509JP-R521JP:~$ docker run -it --rm abazshoushtari/cc-ta-hw2docker-ctf:latest /bin/sh
/ # ls
bin  dev  etc  home  lib  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
/ # ls /home
/ # cd home
/home # ls
/home #
```

فاز اول)

در این فاز از ما خواسته شده تا سه کانتینر الستیک و کیبانا و ردیس را با استفاده از داکر کامپوز بالا بیاوریم و سپس کد بنویسیم که اسم یک فیلم را از کاربر بگیرد و ابتدا در ردیس دنبالی پاسخ بگردد و در صورت عدم پیدا کردن در الستیک و در غیر اینصورت از ای پی ای استفاده کند. همچنین در هر مرحله در صورت نبود به ردیس پاسخ را اضافه کند.

برای این کار ابتدا دیتای داده شده در کورسز را به الستیک با قطعه کد زیر اضافه کردیم.

docker\_1 > my\_image >  insert\_data.py > ...

```
1  from elasticsearch import Elasticsearch
2  import json
3
4  es = Elasticsearch([
5      [{"host": 'elasticsearch', "port": 9200, "scheme": "https"}],
6      timeout=30,
7      max_retries=10,
8      retry_on_timeout=True
9  ])
10 index_schema = {
11     "Poster_Link": {"type": "keyword", "null_value": "null"},
12     "Series_Title": {"type": "keyword", "null_value": "null"},
13     "Released_Year": {"type": "keyword", "null_value": "FALSE"},
14     "Certificate": {"type": "keyword", "null_value": "null"},
15     "Runtime": {"type": "keyword", "null_value": "null"},
16     "Genre": {"type": "keyword", "null_value": "null"},
17     "IMDB_Rating": {"type": "keyword", "null_value": "null"},
18     "Overview": {"type": "keyword", "null_value": "null"},
19     "Meta_score": {"type": "keyword", "null_value": "null"},
20     "Director": {"type": "keyword", "null_value": "null"},
21     "Star1": {"type": "keyword", "null_value": "null"},
22     "Star2": {"type": "keyword", "null_value": "null"},
23     "Star3": {"type": "keyword", "null_value": "null"},
24     "Star4": {"type": "keyword", "null_value": "null"},
25     "No_of_Votes": {"type": "keyword", "null_value": "null"},
26     "Gross": {"type": "keyword", "null_value": "null"},
27 }
28
29 index_settings = {
30     "settings": {
31         "number_of_shards": 1,
32         "number_of_replicas": 1,
33         "analysis": {
34             "analyzer": {
35                 "rebuilt_persian": {
36                     "tokenizer": "standard",
37                     "filter": [
38                         "lowercase",
39                         "decimal_digit",
40                         "persian_normalization",
41                     ]
42                 }
43             }
44         }
```

```

    },
    "mappings": {
        "properties": {}
    }
}

for field_name, config in index_schema.items():
    index_settings["mappings"]["properties"][field_name] = {}
    for cfg_name, cfg_value in index_schema[field_name].items():
        if cfg_name != "type":
            index_settings["mappings"]["properties"][field_name][cfg_name] = cfg_value

es.indices.create(index='movies', ignore=400, body=index_settings)

with open('./movies.json') as f:
    lines = f.readlines()

data = [json.loads(line) for line in lines]

for index, record in enumerate(data):
    if index % 2 == 1:
        es.index(index='movies', body=record)

```

سپس کد پایتون خود را به صورت یک endpoint و با استفاده از فلسک نوشتیم:



```

from elasticsearch import Elasticsearch
from flask import Flask, jsonify, request as request_flask
from walrus import Database
import requests
import json

app = Flask(__name__)

@app.route('/', methods=['GET'])
def movies_system():
    movie_name = request_flask.args.get('movie_name')
    if not movie_name:
        return jsonify({'error': 'لطفاً اسم فیلم خود را ارسال کنید'}), 400

    # Connect to Redis
    db = Database('127.0.0.1', "6379", "0")
    movies_redis_cache = db.cache("movies_redis")
    cache_data = movies_redis_cache.get(movie_name)

    if cache_data is not None:
        correct_data = json.loads(cache_data)
        correct_data['type'] = 'REDIS'
        return jsonify(correct_data)

    # Connect to Elasticsearch
    es = Elasticsearch("https://localhost:9200", http_auth=('elastic', '12345'), verify_certs=False)

    query = {
        "query": {
            "bool": {
                "must": [
                    {"match": {"Series_Title": movie_name}}
                ]
            }
        }
    }

    # headers = {"Content-Type": "application/json"}
    search_result = es.search(index="movies", body=query)

    if len(search_result['hits']['hits']) > 0:
        movies_redis_cache.set(key=movie_name, timeout=3600, value=json.dumps(search_result['hits']['hits'][0]['_source']))
        search_result['hits']['hits'][0]['_source']['type'] = 'ELASTICSEARCH'
        return jsonify(search_result['hits']['hits'][0]['_source'])

    # Search from API if not found in Redis or Elasticsearch
    url = "https://movies-tv-shows-database.p.rapidapi.com/"
    querystring = {"title": movie_name}

```

```

headers = {
    "Type": "get-movies-by-title",
    "X-RapidAPI-Key": "498ca46fd6msh1cfa36a77c07clapledd88jsn788793953a72",
    "X-RapidAPI-Host": "movies-tv-shows-database.p.rapidapi.com"
}
response = requests.get(url, headers=headers, params=querystring)

if response.json()['search_results'] == 0:
    return jsonify({"result": f"اطلاعات فیلم با نام {movie_name} وجود ندارد"})

querystring = {"movieid": response.json()["movie_results"][0]['imdb_id']}
headers = {
    "Type": "get-movie-details",
    "X-RapidAPI-Key": "498ca46fd6msh1cfa36a77c07clapledd88jsn788793953a72",
    "X-RapidAPI-Host": "movies-tv-shows-database.p.rapidapi.com"
}
response = requests.get(url, headers=headers, params=querystring)
api_response = response.json()
del api_response['tagline']
del api_response['imdb_id']
del api_response['status']
del api_response['status_message']
del api_response['youtube_trailer_key']
movies_redis_cache.set(key=movie_name, timeout=3600, value=json.dumps(api_response))
api_response['type'] = 'API'
return jsonify(api_response)

if __name__ == '__main__':
    app.run(host='0.0.0.0', debug=True, port=8000)

```

حالا باید برای کد خودمون یک داکر فایل مینوشتیم تا کد خودمون رو تبدیل به یک ایمج کنیم:

```

FROM python:3.8.10

WORKDIR /app
COPY . /app

RUN pip install -r requirements.txt

COPY run.sh /app/run.sh
RUN chmod +x /app/run.sh

CMD ["/bin/bash", "/app/run.sh"]

```

```

requirements.txt x
docker_1 > my_image > requirements.txt
1  requests
2  elasticsearch == 7.0.0
3  walrus
4  flask
5
6

```

```
run.sh x
docker_1 > my_image > run.sh
1  #!/bin/bash
2  python insert_data.py
3  python app.py
4
```

(فایل نوشته شده برای اینکه ابتدا دیتا در الستیک اضافه شود و سپس فلسک اجرا شود)


ساختن ایمیج:


```
yasaman@yasaman-VivoBook-ASUSLaptop-X509JP-R521JP:~/docker_project$ docker build --network=host -t movie_image /home/yasaman/docker_project/docker_1/my_image
[+] Building 21.0s (11/11) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 206B
=> [internal] load metadata for docker.io/library/python:3.8.10
=> [internal] load .dockerignore
=> transferring context: 2B
=> [1/6] FROM docker.io/library/python:3.8.10
=> [internal] load build context
=> => transferring context: 7.60kB
=> CACHED [2/6] WORKDIR /app
=> [3/6] COPY . /app
=> [4/6] RUN pip install -r requirements.txt
=> [5/6] COPY run.sh /app/run.sh
=> [6/6] RUN chmod +x /app/run.sh
=> exporting to image
=> => exporting layers
=> writing image sha256:1de8156ba21ea40dfba9fbd90f9f4adfacc7cad30cb166fa9f12482abe69d6751
=> naming to docker.io/library/movie_image

View build details: docker-desktop://dashboard/build/default/default/jx5kqng86vus8x0ittxkbajgo

What's Next?
1. Sign in to your Docker account -> docker login
2. View a summary of image vulnerabilities and recommendations -> docker scout quickview
yasaman@yasaman-VivoBook-ASUSLaptop-X509JP-R521JP:~/docker_project$
```

داکر کامپوز نوشته شده:

 docker-compose.yml X

docker\_1 >  docker-compose.yml

```
1  version: '3'
2  services:
3    movie_app:
4      image: movie_image
5      container_name: movie_app
6      restart: always
7      ports:
8        - "8000:8000"
9      volumes:
10       - movie-app-data-volume:/data
11     networks:
12       - elk
13   elasticsearch:
14     image: docker.elastic.co/elasticsearch/elasticsearch:7.4.0
15     container_name: elasticsearch
16     restart: always
17     environment:
18       - xpack.security.enabled=false
19       - xpack.security.transport.ssl.enabled=false
20       - xpack.security.http.ssl.enabled=false
21       - discovery.type=single-node
22       - "ELASTIC_PASSWORD=123456"
23     volumes:
24       - elasticsearch-data-volume:/usr/share/elasticsearch/data
25     ports:
26       - "9200:9200"
27     networks:
28       - elk
29   kibana:
30     image: docker.elastic.co/kibana/kibana:7.4.0
31     container_name: kibana
32     restart: always
33     environment:
34       - ELASTICSEARCH_HOSTS=http://elasticsearch:9200
35       - ELASTICSEARCH_USERNAME=elastic
36       - ELASTICSEARCH_PASSWORD="123456"
37     ports:
38       - "5601:5601"
39     networks:
40       - elk
41     depends_on:
42       - elasticsearch
43   redis:
44     image: redis:latest
45     container_name: redis
46     volumes:
47       - redis-data-volume:/data
48     ports:
```

```
ports:
  - "6379:6379"
networks:
  - elk
```

```
networks:
  elk:
    driver: bridge
```

```
volumes:
  elasticsearch-data-volume:
    driver: local
  redis-data-volume:
    driver: local
  movie-app-data-volume:
    driver: local
```

همانطور که در شکل مشخص است برای الستیک و کیبانا و ردیس و اپ خودمون یک شبکه و ویوم های مختلف درست کردیم برای بررسی شبکه بعد از اجرا داکر کامپوز داریم:

```
yasaman@yasaman-VivoBook-ASUSLaptop-X509JP-R521JP:~$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                                NAMES
c94587164971   docker.elastic.co/kibana/kibana:7.4.0  "/usr/local/bin/dumb..." 39 minutes ago Up 14 seconds 0.0.0.0:5601->5601/tcp, :::5601->5601/tcp kibana
58ae031fd08a   docker.elastic.co/elasticsearch/elasticsearch:7.4.0  "/usr/local/bin/dock..." 39 minutes ago Up 14 seconds 0.0.0.0:9200->9200/tcp, :::9200->9200/tcp, 9300/tcp elasticsearch
334638af0d47   movie_image                           "/bin/bash /app/run..." 2 hours ago   Up 39 minutes 0.0.0.0:8000->8000/tcp, :::8000->8000/tcp movie_app
29904f0b3821   redis:latest                           "docker-entrypoint.s..." 40 hours ago  Up 14 seconds 0.0.0.0:6379->6379/tcp, :::6379->6379/tcp redis
```

```
yasaman@yasaman-VivoBook-ASUSLaptop-X509JP-R521JP:~$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                                NAMES
c94587164971   docker.elastic.co/kibana/kibana:7.4.0  "/usr/local/bin/dumb..." 39 minutes ago Up 14 seconds 0.0.0.0:5601->5601/tcp, :::5601->5601/tcp kibana
58ae031fd08a   docker.elastic.co/elasticsearch/elasticsearch:7.4.0  "/usr/local/bin/dock..." 39 minutes ago Up 14 seconds 0.0.0.0:9200->9200/tcp, :::9200->9200/tcp, 9300/tcp elasticsearch
334638af0d47   movie_image                           "/bin/bash /app/run..." 2 hours ago   Up 39 minutes 0.0.0.0:8000->8000/tcp, :::8000->8000/tcp movie_app
29904f0b3821   redis:latest                           "docker-entrypoint.s..." 40 hours ago  Up 14 seconds 0.0.0.0:6379->6379/tcp, :::6379->6379/tcp redis
```

```
yasaman@yasaman-VivoBook-ASUSLaptop-X509JP-R521JP:~$ docker network inspect docker_1_elk
[
  {
    "Name": "docker_1_elk",
    "Id": "2d5f9d55f9761cc4539d5f23b36dab8421c944f1eeb6271d863b3af4b695c8c9",
    "Created": "2024-04-10T12:03:25.624748687+03:30",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": [
        {
          "Subnet": "172.20.0.0/16",
          "Gateway": "172.20.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": true,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {
      "29904f0b3821eac69aa0e3d28eaa9dd42cdd96dfe1cbd92e516a5c7411f3a4f3": {
        "Name": "redis",
        "EndpointID": "6f92289c63b1d222e3cd633292c9c51430946670a433bbb7abc3fea659dd24de",
        "MacAddress": "02:42:ac:14:00:04",
        "IPv4Address": "172.20.0.4/16",
        "IPv6Address": ""
      },
      "334638af0d4742fdd021412bea8427e8877db72e9c3ad6a507f4687f55128fe2": {
        "Name": "movie_app",
        "EndpointID": "99ed6459b171cbc392d2c5460c9013d81956d89245d9ad0c98880f1931893dc9",
        "MacAddress": "02:42:ac:14:00:02",
        "IPv4Address": "172.20.0.2/16",
        "IPv6Address": ""
      },
      "58ae031fd08ae92cf6c6ac5e8fc7bd93c831d24393a40d3242d13ce18b6cbfe8": {
        "Name": "elasticsearch",
        "EndpointID": "4a1f432cebbd30bc222c1fa8814797650055e98d0fff55907d83bfe262ab2c37",
        "MacAddress": "02:42:ac:14:00:03",
        "IPv4Address": "172.20.0.3/16",
        "IPv6Address": ""
      },
      "c94587164971561c793c5aa62e986e4b0342be7962df5e7ab9352add6c23c578": {
        "Name": "kibana",
        "EndpointID": "7bc8f7cb0cbe06d4ae33a21b3450f5dbba4f28e84cfd807a4edd368fd2494280",
        "MacAddress": "02:42:ac:14:00:05",
        "IPv4Address": "172.20.0.5/16"
      }
    }
  }
]
```

```

    "ConfigOnly": false,
    "Containers": {
      "29904f0b3821eac69aa0e3d28eaa9dd42cdd96dfe1cbd92e516a5c7411f3a4f3": {
        "Name": "redis",
        "EndpointID": "6f92289c63b1d222e3cd633292c9c51430946670a433bbb7abc3fea659dd24de",
        "MacAddress": "02:42:ac:14:00:04",
        "IPv4Address": "172.20.0.4/16",
        "IPv6Address": ""
      },
      "334638af0d4742fdd021412bea8427e8877db72e9c3ad6a507f4687f55128fe2": {
        "Name": "movie_app",
        "EndpointID": "99ed6459b171cbc392d2c5460c9013d81956d89245d9ad0c98880f1931893dc9",
        "MacAddress": "02:42:ac:14:00:02",
        "IPv4Address": "172.20.0.2/16",
        "IPv6Address": ""
      },
      "58ae031fd08ae92cf6c6ac5e8fc7bd93c831d24393a40d3242d13ce18b6cbfe8": {
        "Name": "elasticsearch",
        "EndpointID": "4a1f432cebbd30bc222c1fa8814797650055e98d0fff55907d83bfe262ab2c37",
        "MacAddress": "02:42:ac:14:00:03",
        "IPv4Address": "172.20.0.3/16",
        "IPv6Address": ""
      },
      "c94587164971561c793c5aa62e986e4b0342be7962df5e7ab9352add6c23c578": {
        "Name": "kibana",
        "EndpointID": "7bc8f7cb0cbe06d4ae33a21b3450f5dbba4f28e84cfd807a4edd368fd2494280",
        "MacAddress": "02:42:ac:14:00:05",
        "IPv4Address": "172.20.0.5/16",
        "IPv6Address": ""
      }
    },
    "Options": {},
    "Labels": {
      "com.docker.compose.network": "elk",
      "com.docker.compose.project": "docker_1"
    }
  }
}

```

```

yasaman@yasaman-VivoBook-ASUSLaptop-X509JP-R521JP:~$ docker volume ls
DRIVER      VOLUME NAME
local       51c6a479c9217b9912d54ec02897bd59a887b89ae61ca4216b07b006c0aec61e
local       docker_1_elasticsearch-data-volume
local       docker_1_movie-app-data-volume
local       docker_1_redis-data-volume
yasaman@yasaman-VivoBook-ASUSLaptop-X509JP-R521JP:~$ █

```

خروجی الستیک:

