دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

گزارش پروژه نهایی درس رایانش ابری

دانشجویان:

مرتضی صفری (۹۸۳۱۰۳۹)

نیما محمدی (۹۸۳۱۰۵۷)

استاد: دکتر سید احمد جوادی

بهمن ماه ۱۴۰۲

بسم الله الرحمن الرحیم

گام اول:

انجام شد. اما نوشتن گزارشی در صورت پروژه ذکر نشد.

گام دوم:

Build کردن ایمیج با استفاده از Dockerfile ساخته شده:

docker build -t ftest1:1.0 .

docker run --name app-container --network mynetwork -p 5000:5000 -d ftest1:1.0

```
Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
PS C:\Users\Morteza Sf\Desktop\final_project> docker build -t ftest1:1.0 .
[+] Building 371.7s (14/14) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 937B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/python:3.9-alpine
 => [auth] library/python:pull token for registry-1.docker.io
 => [internal] load build context
 => => transferring context: 1.29kB
 => CACHED [build 1/4] FROM docker.io/library/python:3.9-alpine@sha256:01d2fa38b34f2f79d39882698965e90a3c79e0dc61af39518b386c042fd3f8f0
 => [stage-1 2/5] RUN apk add --no-cache libstdc++
 => [build 2/4] RUN apk add --no-cache build-base
 => [build 3/4] COPY requirements.txt .
 => [build 4/4] RUN pip install --user --no-cache-dir -r requirements.txt
 => [stage-1 3/5] COPY --from=build /root/.local /root/.local
 => [stage-1 4/5] COPY . /app
 => [stage-1 5/5] WORKDIR /app
 => exporting to image
 => => exporting layers
 => => writing image sha256:d51b053c306ba8c9d490422ae7570c88b755b6af641293fb8368a43040c1adba
 => => naming to docker.io/library/ftest1:1.0

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
PS C:\Users\Morteza Sf\Desktop\final_project> docker run --name app-container --network mynetwork -p 5000:5000 -d ftest1:1.0
8766292cc20dffa881389f20556eca225b1a4a4608d135245f66bd3be50423f4
PS C:\Users\Morteza Sf\Desktop\final_project>
```

ارسال ایمیج ساخته شده بر روی داکرهاب و نتیجه آن:

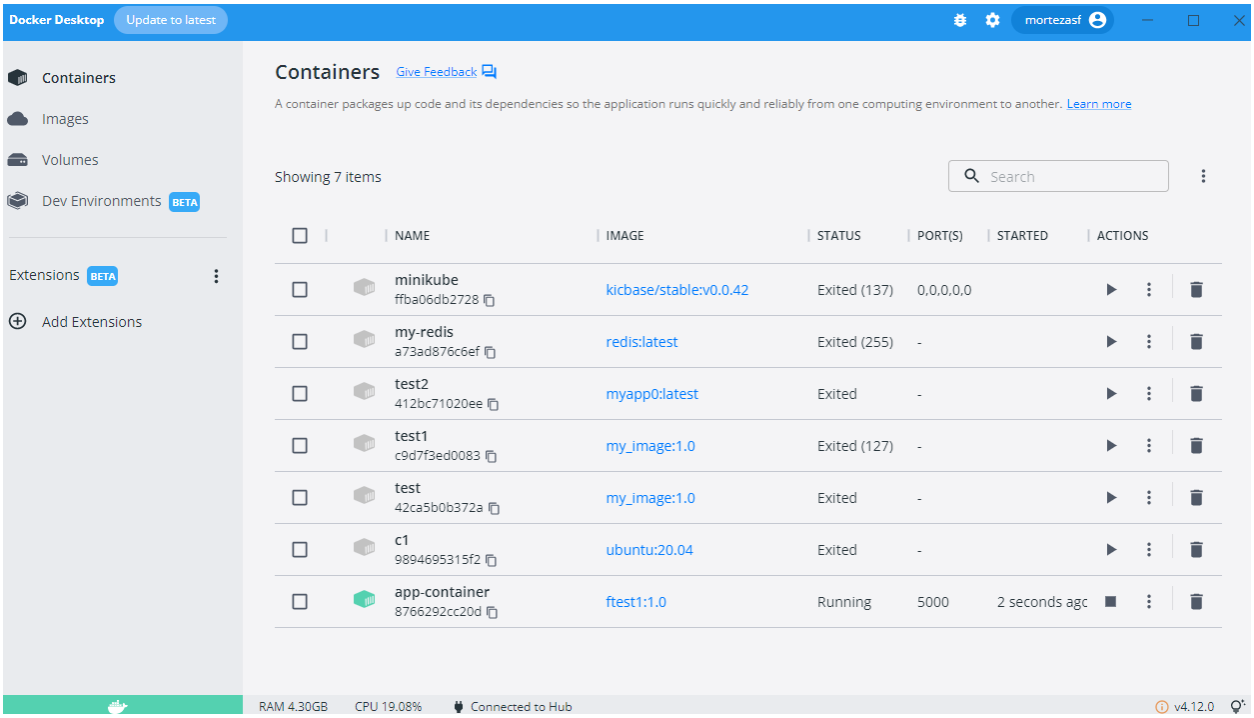
docker tag ftest1:1.0 mortezasf/ftest1:1.0

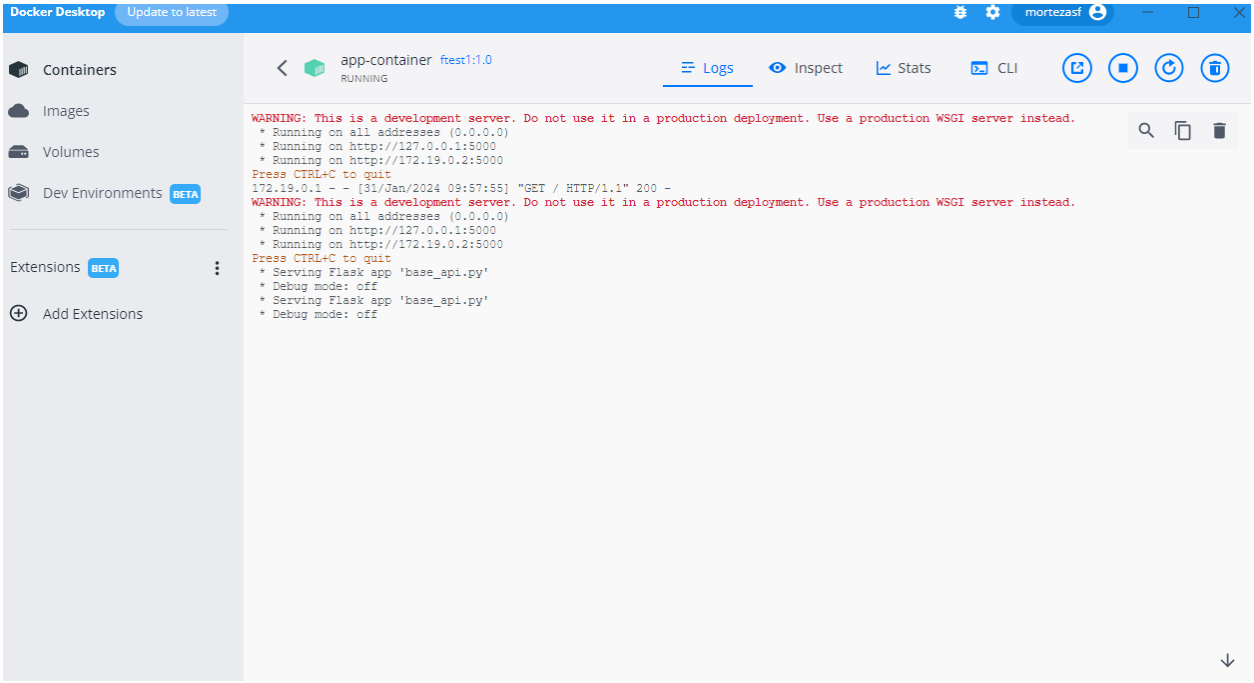
docker push mortezasf/ftest1:1.0

```
PS C:\Users\Morteza Sf\Desktop\final_project> docker login
Authenticating with existing credentials...
Login Succeeded

Logging in with your password grants your terminal complete access to your account.
For better security, log in with a limited-privilege personal access token. Learn more at https://docs.docker.com/go/access-tokens/
PS C:\Users\Morteza Sf\Desktop\final_project> docker tag ftest1:1.0 mortezasf/ftest1:1.0
PS C:\Users\Morteza Sf\Desktop\final_project> docker push mortezasf/ftest1:1.0
The push refers to repository [docker.io/mortezasf/ftest1]
5f70bf18a086: Mounted from mortezasf/myapp
49e41e354665: Pushed
3e813464c5bd: Pushed
704206ffa6d2: Pushed
cdfe7f245c06: Mounted from library/python
b8d25f9fcffb: Mounted from library/python
0bda99f2fb62: Mounted from library/python
678cac8b069e: Mounted from library/python
d4fc045c9e3a: Mounted from library/python
1.0: digest: sha256:8003958b80c20d1fbd2e3d0efd5bc072d8458f86831610d131e7816127a032dc size: 2204
PS C:\Users\Morteza Sf\Desktop\final_project>
```

در صورتی که پروژه خود را با استفاده از ایمیج ساخته شده بر روی سیستم شخصی خود تست کردید، تصاویر مربوطه را قرار دهید (این مرحله اجباری نیست ولی توصیه میشود)





127.0.0.1:5000/api/all


```
[
  {
    "address": "216.239.38.120",
    "created_at": "1706610291.3625097",
    "failure": 0,
    "id": "1",
    "last_failure": "null",
    "success": 19
  },
  {
    "address": "www.google.com",
    "created_at": "1706610317.3691113",
    "failure": 0,
    "id": "2",
    "last_failure": "null",
    "success": 19
  },
  {
    "address": "192.168.1.2",
    "created_at": "1706611395.4748158",
    "failure": 12,
    "id": "3",
    "last_failure": "1706637480.6508563",
    "success": 0
  }
]
```


صحت اجرا شدن داکرکامپوز و مراحل آن:

```
PS C:\Users\Morteza Sf\Desktop\final_project> docker-compose up --build
[+] Building 4.1s (13/14)
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 32B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/python:3.9-alpine
 => [auth] library/python:pull token for registry-1.docker.io
 => [internal] load build context
 => => transferring context: 10.62kB
 => [build 1/4] FROM docker.io/library/python:3.9-alpine@sha256:01d2fa38b34f2f79d39882698965e90a3c79e0dc61af39518b386c042fd3f8f0
 => CACHED [stage-1 2/5] RUN apk add --no-cache libstdc++
 => CACHED [build 2/4] RUN apk add --no-cache build-base
 => CACHED [build 3/4] COPY requirements.txt .
 => CACHED [build 4/4] RUN pip install --no-cache-dir -r requirements.txt --user
 => CACHED [stage-1 3/5] COPY --from=build /root/.local /root/.local
 => [stage-1 4/5] COPY . /app
 => [stage-1 5/5] WORKDIR /app
 => exporting to image
 => => exporting layers
 => => writing image sha256:c853b0b21de120cf90ab9b6f72c5f31578dfe3cf5ae5cb1c853d6e482990d9aa
 => => naming to docker.io/library/final_project-api

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
[+] Running 1/1
 - Container final_project-api-1  Recreated
Attaching to final_project-api-1
final_project-api-1  |  * Serving Flask app 'base_api.py'
final_project-api-1  |  * Debug mode: off
final_project-api-1  | WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
final_project-api-1  |  * Running on all addresses (0.0.0.0)
final_project-api-1  |  * Running on http://127.0.0.1:5000
final_project-api-1  |  * Running on http://172.24.0.2:5000
final_project-api-1  | Press CTRL+C to quit
final_project-api-1  | 172.24.0.1 - - [31/Jan/2024 22:46:37] "GET / HTTP/1.1" 200 -
```

گام سوم:

با استفاده از دستور get kubectl صحت ایجاد منابع بر روی کلاستر را نمایش دهید:

```
PS C:\Users\Morteza Sf\Desktop\final_project> kubectl apply -f db-credentials.yaml
Error from server (BadRequest): error when creating "db-credentials.yaml": Secret in version "v1" cannot be handled as a
pe []uint8
PS C:\Users\Morteza Sf\Desktop\final_project> kubectl apply -f db-credentials.yaml
Error from server (BadRequest): error when creating "db-credentials.yaml": Secret in version "v1" cannot be handled as a
pe []uint8
PS C:\Users\Morteza Sf\Desktop\final_project> kubectl apply -f db-credentials.yaml
secret/db-credentials created
PS C:\Users\Morteza Sf\Desktop\final_project> kubectl get pods
NAME                      READY   STATUS    RESTARTS   AGE
my-app-584d94d9f8-c42r4   1/1     Running   0          2m36s
my-app-584d94d9f8-l4wcc   1/1     Running   0          2m36s
PS C:\Users\Morteza Sf\Desktop\final_project>
```

آدرس IP پادها و نحوه برقراری ارتباط میان آنها و سرویس ساخته شده:

kubectl get pods -o wide

```
PS C:\Users\Morteza Sf\Desktop\final_project> kubectl get pods -o wide
NAME                      READY   STATUS    RESTARTS   AGE     IP           NODE       NOMINATED NODE   READINESS GATES
my-app-584d94d9f8-c42r4   1/1     Running   0          6m50s   10.244.0.52  minikube   <none>           <none>
my-app-584d94d9f8-l4wcc   1/1     Running   0          6m50s   10.244.0.53  minikube   <none>           <none>
PS C:\Users\Morteza Sf\Desktop\final_project>
```

برای statefulset مربوط به دیتابیس چه تعداد پاد ایجاد کردید؟ دلیل کار خود را توضیح دهید:

در StatefulSet مربوط به دیتابیس، ما تعداد ۲ پاد (replica) ایجاد کرده‌ایم با استفاده از فیلد replicas: 2 در بخش spec این.StatefulSet

ایجاد ۲ پاد در StatefulSet به دلیل اهمیت بالای داده‌های دیتابیس و نیاز به پایداری و انعطاف‌پذیری است. با ایجاد ۲ پاد، هر پاد دارای یک نسخه مستقل از دیتابیس است که به صورت پایدار و قابلیت بازیابی در صورت خرابی ارائه می‌شود.

این رویکرد برای سیستم‌های دیتابیسی مانند MySQL که نیازمند حفظ اطلاعات و پایداری برای دسترسی مستمر به داده هستند، مناسب است. با توزیع بار بین پادها و ایجاد نسخه‌های مستقل، هر پاد قادر به پیشبرد کارکرد دیتابیس است و در صورت خرابی یکی از پادها، پاد دیگری می‌تواند به عنوان جایگزین فعال شود. علاوه بر این، با استفاده از VolumeClaimTemplates و اختصاص فضای ذخیره‌سازی مجزا برای هر پاد، هر پاد دارای فضای ذخیره‌سازی مجزا برای داده‌های دیتابیس خود می‌باشد، که این امر از اهمیت بالایی برخوردار است زیرا هر پاد می‌تواند داده‌های خود را به صورت مستقل ذخیره و به آنها دسترسی داشته باشد. به طور کلی، استفاده از StatefulSet برای دیتابیس‌ها بهترین رویکرد است و امکاناتی را برای مدیریت پایدار و قابل بازیابی داده‌های دیتابیس در محیط Kubernetes فراهم می‌کند.

نحوه استفاده از سرویس مستر و رپلیکاها:

درKubernetes ، سرویس (Service) و رپلیکا (Replica) دو مفهوم مهم برای مدیریت ارتباط بین بخش‌های مختلف برنامه و توزیع بار می‌باشند. در زیر نحوه استفاده از هر یک را توضیح می‌دهم:

سرویس:(Service)

سرویس در Kubernetes یک نقطه ورود (Entry Point) ثابت برای ارتباط با پادها است. سرویس می‌تواند به عنوان یک لایه تنظیم بار (Load Balancer) عمل کند و درخواست‌ها را به پادهای مختلف توزیع کند. برای استفاده از سرویس، شما یک سرویس را تعریف کرده و به پادها (Replicas) مربوطه ارتباط می‌دهید. سپس می‌توانید با استفاده از نام سرویس و پورت‌های مربوطه، از هر جایی درون یا خارج از خوشه Kubernetes به پادها دسترسی پیدا کنید.

رپلیکا:(Replica)

رپلیکا در Kubernetes به تعداد کپی‌های یک پاد گفته می‌شود. با افزایش تعداد رپلیکا، بخشی از بار کاری را بر روی پادهای بیشتر توزیع می‌کنید. برای استفاده از رپلیکا، شما یک Deployment یا StatefulSet تعریف می‌کنید که تعداد خاصی از رپلیکاها را مشخص می‌کند. Kubernetes سپس می‌تواند رپلیکاهای اضافی را ایجاد کند و به صورت خودکار توزیع بار را بین آنها انجام دهد.

```
PS C:\Users\Morteza Sf\Desktop\final_project> kubectl get svc
NAME             TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
database         ClusterIP      10.111.88.20     <none>        10640/TCP      108s
kubernetes       ClusterIP      10.96.0.1        <none>        443/TCP        47d
my-app-service   ClusterIP      10.108.193.36    <none>        5000/TCP       86m
my-redis         ClusterIP      10.107.237.162   <none>        6379/TCP       47d
server-service   LoadBalancer   10.107.88.115    <pending>     80:30718/TCP   47d
PS C:\Users\Morteza Sf\Desktop\final_project> minikube service my-app-service
|-----------|----------------|-------------|--------------|
| NAMESPACE |      NAME      | TARGET PORT |     URL      |
|-----------|----------------|-------------|--------------|
| default   | my-app-service |             | No node port |
|-----------|----------------|-------------|--------------|
* service default/my-app-service has no node port
* Starting tunnel for service my-app-service.
|-----------|----------------|-------------|------------------------|
| NAMESPACE |      NAME      | TARGET PORT |          URL           |
|-----------|----------------|-------------|------------------------|
| default   | my-app-service |             | http://127.0.0.1:64908 |
|-----------|----------------|-------------|------------------------|
* Opening service default/my-app-service in default browser...
! Because you are using a Docker driver on windows, the terminal needs to be open to run it.
```

127.0.0.1:64908/api/all


```
[
  {
    "address": "216.239.38.120",
    "created_at": "1706610291.3625097",
    "failure": 0,
    "id": "1",
    "last_failure": "null",
    "success": 19
  },
  {
    "address": "www.google.com",
    "created_at": "1706610317.3691113",
    "failure": 0,
    "id": "2",
    "last_failure": "null",
    "success": 19
  },
  {
    "address": "192.168.1.2",
    "created_at": "1706611395.4748158",
    "failure": 12,
    "id": "3",
    "last_failure": "1706637480.6508563",
    "success": 0
  }
]
```

اجرا و تست با استفاده از port forwarding:

```
PS C:\Users\Morteza Sf\Desktop\final_project> kubectl get services
NAME             TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
database         ClusterIP      10.111.88.20     <none>        10640/TCP      11m
kubernetes       ClusterIP      10.96.0.1        <none>        443/TCP        47d
my-app-service   ClusterIP      10.108.193.36    <none>        5000/TCP       96m
my-redis         ClusterIP      10.107.237.162   <none>        6379/TCP       47d
server-service   LoadBalancer   10.107.88.115    <pending>     80:30718/TCP   47d
PS C:\Users\Morteza Sf\Desktop\final_project> kubectl port-forward service/my-app-service 8888:5000
Forwarding from 127.0.0.1:8888 -> 5000
Forwarding from [::1]:8888 -> 5000
Handling connection for 8888
Handling connection for 8888
Handling connection for 8888
Handling connection for 8888
Handling connection for 8888
Handling connection for 8888
Handling connection for 8888
```

localhost:8888/api/all


```
[
  {
    "address": "216.239.38.120",
    "created_at": "1706610291.3625097",
    "failure": 0,
    "id": "1",
    "last_failure": "null",
    "success": 19
  },
  {
    "address": "www.google.com",
    "created_at": "1706610317.3691113",
    "failure": 0,
    "id": "2",
    "last_failure": "null",
    "success": 19
  },
  {
    "address": "192.168.1.2",
    "created_at": "1706611395.4748158",
    "failure": 12,
    "id": "3",
    "last_failure": "1706637480.6508563",
    "success": 0
  }
]
```