دانشگاه صنعتی امیرکبیر
(پلیتکنیک تهران)
دانشکده مهندسی کامپیوتر

درس رایانش ابری
پروژه پایانی

(گروه ۳۰)

امیرمهدی زرین‌نژاد ۹۷۳۱۰۸۷

محمدعرفان قاسمی ۹۷۳۱۰۴۸

نیم‌سال دوم ۱۴۰۰

گام دوم

۱) build کردن ایمیج با استفاده از Dockerfile ساخته شده

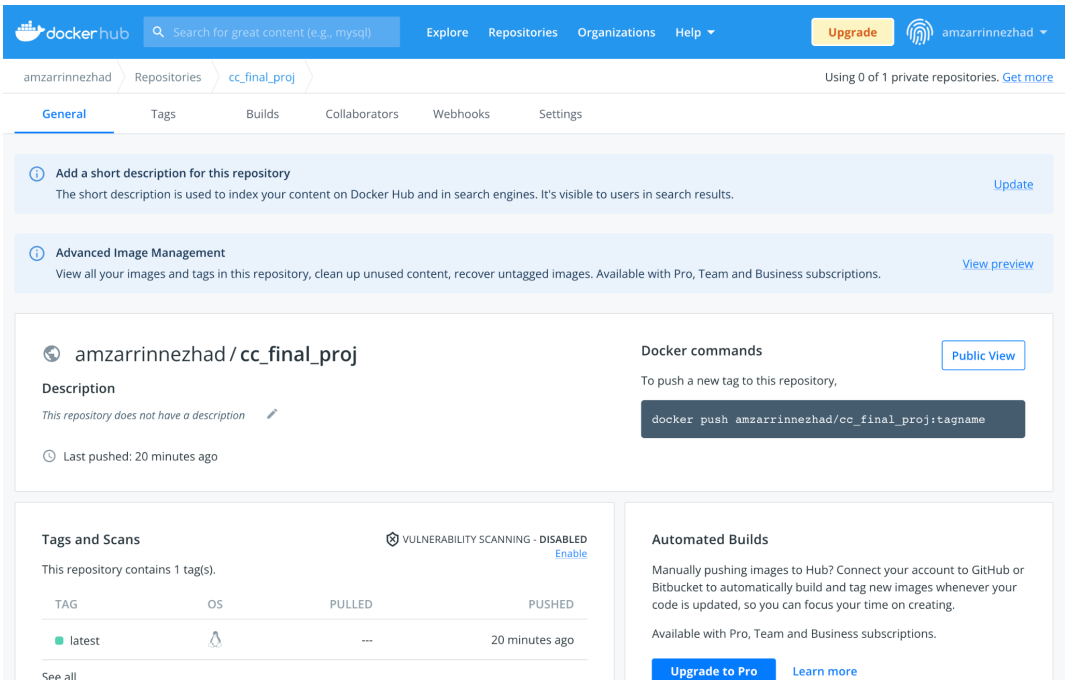
```
➜  Cloud-Computing-Final-Project-main docker build -f /Users/apple/PycharmProjects/Term8/Cloud/Cloud-Computing-Final-Project-main/Dockerfile . -t cc_final_proj
[+] Building 22.3s (12/12) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 140B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/python:3.9-alpine
 => [internal] load build context
 => => transferring context: 11.37kB
 => [build 1/4] FROM docker.io/library/python:3.9-alpine@sha256:89ea7c66e4acf3d466a7ba1a3c8cf20895e3d6e69c8f5b0b3ccd6ffaf38075bb
 => CACHED [build 2/4] RUN python -m venv /opt/venv
 => CACHED [target 2/4] WORKDIR /app
 => [build 3/4] COPY requirements.txt .
 => [build 4/4] RUN pip install --no-cache-dir -r requirements.txt
 => [target 3/4] COPY --from=build /opt/venv /opt/venv
 => [target 4/4] COPY /app .
 => exporting to image
 => => exporting layers
 => => writing image sha256:81a3434e4cc7380f2360534853f67e8870ef3671b790be8094cf85cc62e747d5
 => => naming to docker.io/library/cc_final_proj

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
➜  Cloud-Computing-Final-Project-main docker image ls
REPOSITORY                   TAG       IMAGE ID       CREATED          SIZE
cc_final_proj                latest    81a3434e4cc7   14 seconds ago   70.5MB
```

۲) ارسال ایمیج ساخته شده بر روی داکرهاب و نتیجه آن

```
➜  Cloud-Computing-Final-Project-main docker image tag cc_final_proj amzarrinnezhad/cc_final_proj
➜  Cloud-Computing-Final-Project-main docker image ls
REPOSITORY                        TAG       IMAGE ID       CREATED          SIZE
amzarrinnezhad/cc_final_proj      latest    81a3434e4cc7   4 minutes ago    70.5MB
cc_final_proj                     latest    81a3434e4cc7   4 minutes ago    70.5MB
```

```
➜  Cloud-Computing-Final-Project-main docker push amzarrinnezhad/cc_final_proj
Using default tag: latest
The push refers to repository [docker.io/amzarrinnezhad/cc_final_proj]
a9dbbd663bf6: Pushed
5841103caa85: Pushed
e765b11637c2: Pushed
8774fddab619: Mounted from library/python
57c4901cd8f5: Mounted from library/python
97dd4708fc95: Mounted from library/python
09c126bb3acd: Mounted from library/python
24302eb7d908: Mounted from library/python
latest: digest: sha256:d99e8b5ddd5be4691f285a06b9e7b9d3ac82dae081b5e85d6cb85f5744cbcec6 size: 1994
```



۳) 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)

```
➜  Cloud-Computing-Final-Project-main docker run --name final-test -p 5000:8080 amzarrinnezhad/cc_final_proj
 * Serving Flask app 'app' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000 (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 672-093-480
```

۴) محتویات Dockerfile

```dockerfile
FROM python:3.9-alpine AS build

RUN python -m venv /opt/venv
ENV PATH="/opt/venv/bin:$PATH"

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt


FROM python:3.9-alpine AS target
WORKDIR /app
COPY --from=build /opt/venv /opt/venv
ENV PATH="/opt/venv/bin:$PATH"

ENV ENV_FILE=/env/.env

EXPOSE 8080

COPY /app .
CMD ["python", "app.py"]
```

گام سوم

- اولین کامپوننت مورد نیاز یک ConfigMap براي پروژه است تا بتوان پورت سرور، زمان انقضاي آدرس‌هاي ایجاد شده و آدرس سرور دیتابیس از آن خوانده شود.

- کامپوننت بعدي یک Secret است که وظیفه ذخیره‌سازي اسم و رمز عبور دیتابیس را بر عهده دارد. از آن جایی که این اطلاعات، مخصوصا رمز عبور، جزء اطلاعات حساس هستند باید آنها را در Secret ذخیره نمود.

- به منظور پایدار ماندن اطلاعات دیتابیس در صورت بروز مشکل براي پادهاي مربوطه، لازم است تا براي آن Persistent Volume تعریف کنیم. در نتیجه گام بعدي ایجاد Persistent Volume و در ادامه ساخت Persistent Volume Claim براي استفاده از آن است.

- سپس باید یک توصیف deployment بنویسید که وظیفه آماده‌سازي و نگهداري از دیتابیس را بر عهده دارد (نحوه ایجاد دیتابیس به انتخاب شماست. تنها نکته مهم برخورداري از رمز عبور تعریف شده در توصیف Secret است). فراموش نکنید که Persistent Volume Claim ساخته شده در مرحله قبل را بر این دیپلویمنت سوار کنید.

به نظر شما تعداد پادهای مناسب این توصیف چند عدد است؟

- برای دسترسی به این دیتابیس به یک Service نیاز است که با استفاده از آن می‌توانیم ارتباط پروژه و دیتابیس را قرار کنیم.

- حال میتوان یک Deployment نوشت که وظیفه آماده‌سازي و نگهداري پادها را بر عهده دارد. تعداد replica را برابر با 2 تعیین کنید (دقت داشته باشید که در این توصیف شما باید Secret و ConfigMap ساخته شده را در اختیار پروژه قرار دهید تا مقادیر لازم از طریق آنها پر شود).

- آخرین مورد یک Service است که با استفاده از آن میتوانیم به پروژه و در واقع سروري که توسعه داده‌ایم دسترسی داشته باشیم.

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

kubectl apply:

```
➜  k8s kubectl apply -f cm.yaml
configmap/privatenote-config created
➜  k8s kubectl apply -f secret.yml
secret/redis-secret created
➜  k8s kubectl apply -f pv.yaml
persistentvolume/redis created
➜  k8s kubectl apply -f pvc.yaml
persistentvolumeclaim/redis-pvc created
➜  k8s kubectl apply -f redis-deployment.yaml
deployment.apps/redis created
➜  k8s kubectl apply -f redis-service.yaml
service/redis-service created
➜  k8s kubectl apply -f privatenote-deployment.yaml
deployment.apps/privatenote created
➜  k8s kubectl apply -f privatenote-service.yaml
service/privatenote-service created
➜  k8s
```

kubectl get:

```
➜  k8s kubectl get configmaps
NAME                 DATA   AGE
kube-root-ca.crt     1      71d
privatenote-config   1      6m6s
weather-server       1      62d
➜  k8s kubectl get secrets
NAME                  TYPE                                  DATA   AGE
default-token-9h2kp   kubernetes.io/service-account-token   3      71d
redis-secret          Opaque                                1      5m52s
➜  k8s kubectl get persistentvolumes
NAME    CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM               STORAGECLASS   REASON   AGE
redis   100Mi      RWO            Retain           Bound    default/redis-pvc   manual                  5m35s
➜  k8s kubectl get persistentvolumeclaims
NAME        STATUS   VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   AGE
redis-pvc   Bound    redis    100Mi      RWO            manual         5m40s
➜  k8s kubectl get services
NAME                 TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
kubernetes           ClusterIP   10.96.0.1        <none>        443/TCP    71d
privatenote-service  ClusterIP   10.108.133.233   <none>        80/TCP     16m
redis-service        ClusterIP   10.102.191.124   <none>        6379/TCP   17m
➜  k8s kubectl get deployments.apps
NAME         READY   UP-TO-DATE   AVAILABLE   AGE
privatenote  2/2     2            2           5m1s
redis        1/1     1            1           5m48s
```

```
➜  k8s kubectl get endpoints
NAME                 ENDPOINTS                         AGE
kubernetes           192.168.49.2:8443                 71d
privatenote-service  172.17.0.6:8080,172.17.0.7:8080   14m
redis-service        172.17.0.5:6379                   14m
```

```
➜  k8s kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
privatenote-6cf7ffb98c-4ls56  1/1     Running   0          42s
privatenote-6cf7ffb98c-wkds7  1/1     Running   0          42s
redis-864d84c558-lzf6v        1/1     Running   0          89s
➜  k8s kubectl get pods -o wide
NAME                          READY   STATUS    RESTARTS   AGE    IP           NODE       NOMINATED NODE   READINESS GATES
privatenote-6cf7ffb98c-4ls56  1/1     Running   0          69s    172.17.0.6   minikube   <none>           <none>
privatenote-6cf7ffb98c-wkds7  1/1     Running   0          69s    172.17.0.7   minikube   <none>           <none>
redis-864d84c558-lzf6v        1/1     Running   0          116s   172.17.0.5   minikube   <none>           <none>
➜  k8s
```

```
➜  k8s kubectl get services
NAME                  TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
kubernetes            ClusterIP   10.96.0.1        <none>        443/TCP    71d
privatenote-service   ClusterIP   10.108.133.233   <none>        80/TCP     82s
redis-service         ClusterIP   10.102.191.124   <none>        6379/TCP   113s
```

```
➜  k8s kubectl get deployments.apps -o wide
NAME          READY   UP-TO-DATE   AVAILABLE   AGE     CONTAINERS    IMAGES                  SELECTOR
privatenote   2/2     2            2           95s     privatenote   erfanghasemi/ccp:1.3    app=privatenote
redis         1/1     1            1           2m22s   redis         redis:6.2.6-alpine      app=redis
```

```
➜  k8s kubectl get services privatenote-service
NAME                  TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)   AGE
privatenote-service   ClusterIP   10.108.133.233   <none>        80/TCP    14m
```

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

```
➜  k8s kubectl get pods
NAME                           READY   STATUS    RESTARTS   AGE
privatenote-6cf7ffb98c-4ls56   1/1     Running   0          42s
privatenote-6cf7ffb98c-wkds7   1/1     Running   0          42s
redis-864d84c558-lzf6v         1/1     Running   0          89s
➜  k8s kubectl get pods -o wide
NAME                           READY   STATUS    RESTARTS   AGE    IP           NODE       NOMINATED NODE   READINESS GATES
privatenote-6cf7ffb98c-4ls56   1/1     Running   0          69s    172.17.0.6   minikube   <none>           <none>
privatenote-6cf7ffb98c-wkds7   1/1     Running   0          69s    172.17.0.7   minikube   <none>           <none>
redis-864d84c558-lzf6v         1/1     Running   0          116s   172.17.0.5   minikube   <none>           <none>
➜  k8s
```

```
➜  k8s kubectl get endpoints
NAME                  ENDPOINTS                         AGE
kubernetes            192.168.49.2:8443                 71d
privatenote-service   172.17.0.6:8080,172.17.0.7:8080   14m
redis-service         172.17.0.5:6379                   14m
```

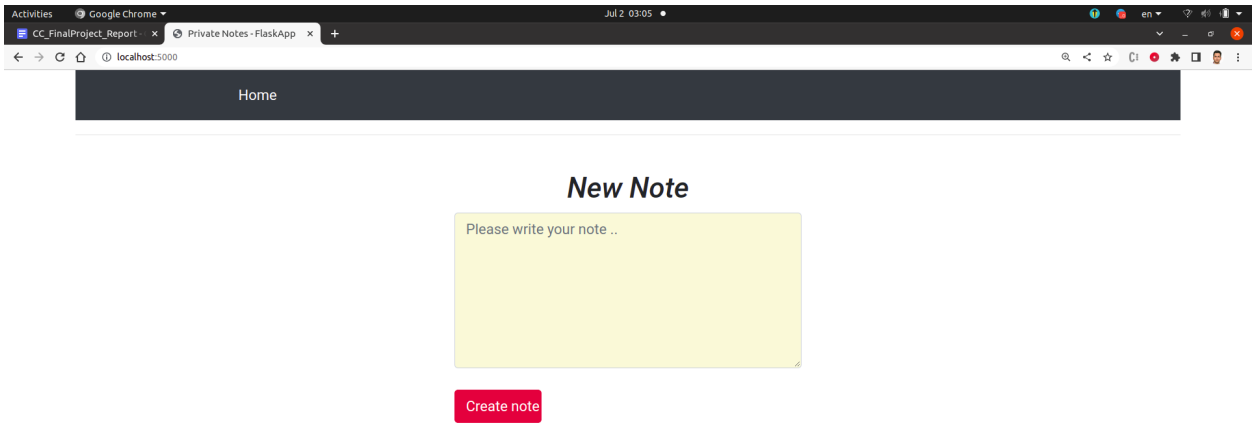
همانطور که مشاهده می‌شود، سرویس مجموعه آدرسهای پادهایش را نگه می دارد تا در صورت جایگزینی و بالا آمدن پاد جدید از همان آدرسها استفاده کند و به اینصورت آدرس‌های پادها را مدیریت کند.

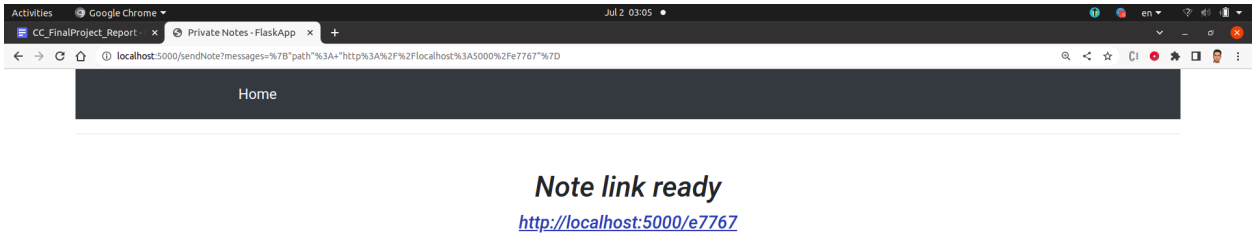
۳) برای دیپلویمنت مربوط به دیتابیس چه تعداد پاد ایجاد کردید؟ دلیل کار خود را توضیح دهید.

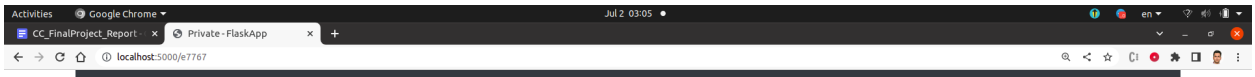
تعداد پادها یا replica را برابر ۱ قرار داده‌ایم، زیرا برای دسترسی به دیتابیس فقط به یک پاد نیاز داریم، همچنین درصورت تعریف کردن بیش از یک پاد در واقع باید کنترل شود که دیتاهای رپلیکاهای مختلف باید با هم سینک و همگام شوند. در واقع زمانی که یک درخواست برای یک پاد مشخص می‌رود و در آن دیتایی ذخیره می‌شود، در واقع پادهای دیگر نیز باید خبر دارد شوند و دیتای جدید به پادهای دیگر هم منتقل شود و در واقع باید پادهای دیگر نیز آپدیت شوند.

آزمون پروژه

```
➜  k8s kubectl port-forward service/privatenote-service 5000:80
Forwarding from 127.0.0.1:5000 -> 8080
Forwarding from [::1]:5000 -> 8080
```





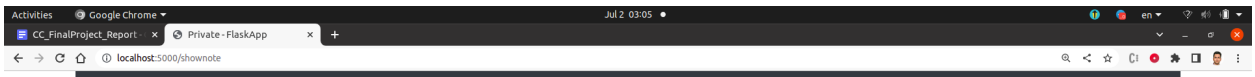


Home

Read and destroy?

You're about to read and detroy the note with id **e7767**.

Yes, show me the note with id:e7767 | No, not now



Home

Your note is here :)

salam In note baraye test hast

موارد امتیازی

الف) ساختن یک کامپوننت HPA در کلاستر کوبرنتیز به منظور انجام عملیات auto scaling

۱) پارامترهای موجود جهت مقیاس کردن خودکار را بیان کنید

به منظور انجام autoscaling توسط کوبرنتیز، نیاز به مشخص کردن معیاری که به کمک آن بار پادها و نیاز به scale up/down مشخص می‌شود داریم. در api نسخه یک، تنها معیار بهره‌وری CPU برای پادها وجود داشت. اما در نسخه‌های بتای فعلی، می‌توان از معیارهای متفاوتی برای این کار استفاده کرد. این معیارها عبارتند از:

1. containerResource: این معیار، مربوط به یک منبع مشخص (مانند cpu و memory) برای یک کانتینر در هر یک از پادها است. این معیار به شکل built in در کوبرنتیز وجود دارد و قابل استفاده است.

2. external: این معیار مربوط به یک معیار سراسری است که به هیچ یک از object های کوبرنتیز مربوط نمی‌شود. به عنوان مثال، اندازه یک صف در یک سرویس پیام‌رسانی ابری در خارج از کلاستر کوبرنتیز، یک معیار external است.

3. object: یک معیار مربوط به یک object در کلاستر کوبرنتیز (مثلا hits-per-second در ingress) است.

4. pods: این معیار، توصیف‌کننده هر یک از پادها در scale هدف فعلی (مثلا packets-per-second) است. در این معیار، مقادیر پیش از مقایسه با میزان هدف (حد آستانه مشخص شده برای scale کردن)، میانگین گرفته می‌شوند.

5. resource: مانند containerResource است. با این تفاوت که برای یک پاد (نه یک کانتینر) مشخص می‌شود.

۲) شما کدامیک از این پارامترها را براي ایجاد HPA استفاده کردید؟ دلیل خود را شرح دهید

در اینجا از معیار resource (و در حالت cpu) برای مقیاس کردن استفاده شده است. از بین موارد مطرح شده، مورد ۲، ۳ و ۴ به تنهایی در این کاربرد در دسترس نبودند. همچنین مورد ۱ و ۵ نیز، با توجه به اینکه در هر یک از پادها تنها یک کانتینر استفاده شده است، تفاوتی ندارد.

۳) دستور و یا توصیف مورد استفاده برای ساخت HPA

```
➜  Cloud-Computing-Final-Project-main kubectl apply -f hpa.yaml
horizontalpodautoscaler.autoscaling/privatenote configured
➜  Cloud-Computing-Final-Project-main kubectl get hpa
NAME          REFERENCE                TARGETS         MINPODS   MAXPODS   REPLICAS   AGE
privatenote   Deployment/privatenote   <unknown>/50%   1         5         2          5m22s
```

k8s > ! hpa.yaml

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: privatenote
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: privatenote
  minReplicas: 1
  maxReplicas: 5
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 50
```

k8s > ! privatenote-deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: privatenote
  name: privatenote
spec:
  replicas: 2
  selector:
    matchLabels:
      app: privatenote
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: privatenote
    spec:
      containers:
      - image: erfanghasemi/ccnote:1.4
        name: privatenote
        resources:
          requests:
            cpu: "40m"
        ports:
          - containerPort: 8080
        env:
          - name: REDIS_SECRET
            valueFrom:
              secretKeyRef:
                key: REDIS_PASSWORD
                name: redis-secret

        volumeMounts:
          - mountPath: /env/.env
            subPath: .env
            name: config-map
            readOnly: true
      volumes:
        - name: config-map
          configMap:
            name: privatenote-config
status: {}
```

ب) اجرای دیتابیس خود با استفاده از توصیف stateful set و جایگزین کردن آن با deployment

۱) دلیل استفاده از stateful set بجای deployment

در واقع StatefulSet یک کامپوننت در کوبرنتیز می‌باشد که برای اپلیکیشن‌های stateful استفاده می‌شود، اپلیکیشن‌های stateful، اپلیکیشن‌هایی هستند که دیتا در خود ذخیره می‌کنند تا بتوانند حالت خود را ذخیره کنند و بتوانند حالت خود را دنبال کنند(trace) در مقابل اپلیکیشن‌های stateless هیچ حالتی از خود را ذخیره نمی‌کنند(do not keep record of state of previous interactions) و هر ریکوست یا interaction به صورت کاملا isolated هندل می‌شود و با استفاده از اطلاعات خود درخواست فعلی هندل می‌شود.

به دلیل تفاوت این دو نوع اپلیکیشن‌ها، دیپلوی کردن این اپلیکیشن‌ها نیز با هم متفاوت است. اپلیکیشن‌های stateless توسط deployment دیپلوی می‌شوند و اپلیکیشن‌های stateful توسط statefulset دیپلوی می‌شوند.

در deployment پادها بصورت رندوم تولید می‌شوند و در انتهای آن‌ها random hashes وجود دارد و همچنین identical و interchangeable هستند و در واقع یک سرویس وجود دارد که کار load balancing را برای هر پاد و برای هر ریکوست انجام می‌دهد و توجه شود که در واقع scaling بصورت رندوم بین پادها انجام می‌شود زیرا پادها identical هستند.

در statefulset نمی‌توان بصورت همزمان و به هر ترتیب دلخواهی پادها را create یا delete کرد و نمی‌توان پادها را بصورت رندوم آدرس‌دهی کرد و این امر به این دلیل است که پادها identical نیستند، در واقع هر کدام Identity مختص به خود را دارند(این امر باعث تفاوت بین deployment و statefulset شده است).

در واقع یک پاد نقش master را دارد و پادهای دیگر همگی slave هستند و master هم می‌تواند بخواند و هم می‌تواند بنویسد اما دیگر پادها که slave هستند فقط می‌تواند بخوانند زیرا در غیر اینصورت data inconsistency پیش می‌آید.

هر زمانی که master دیتای جدیدی write می‌کند، پادهای slave باید حواسشان باشد که دیتای خود را با master همگام کنند و synchronized باشند، همچنین توجه شود که این کار باعث می‌شود که دیگر دغدغه‌ی همگام بودن داده‌ها را نداشته باشیم(زیرا statefulset خودش این را هندل می‌کند)، همچنین توجه شود که هر پاد volume مجزای خود را داراست درصورتی که همگی از یک تعریف statefulset درست شده‌اند اما در deployment صرفا یک volume برای تمامی پادها وجود دارد.

۲) توصیف مورد استفاده برای ساخت stateful set

توصیف configmap:

```yaml
apiVersion: v1
data:
  .env: |
    HOST=0.0.0.0
    PORT=8080
    URL_EX=60
    REDIS_HOST=redis-0.redis
    REDIS_PORT=6379
    REDIS_DB=0
  master.conf: |
    bind 0.0.0.0
    protected-mode yes
    port 6379
    tcp-backlog 511
    timeout 0
    tcp-keepalive 300
    daemonize no
    supervised no
    pidfile /var/run/redis_6379.pid
    loglevel notice
    logfile ""
  slave.conf: |
    slaveof redis-0.redis 6379
kind: ConfigMap
metadata:
  creationTimestamp: null
  name: privatenote-config
```

توصیف redis-service:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: redis
  labels:
    app: redis
spec:
  type: ClusterIP
  clusterIP: None
  selector:
    app: redis
    # role: master
  ports:
    - protocol: TCP
      port: 6379
      targetPort: redis-port
```

توصیف redis-service:

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  creationTimestamp: null
  labels:
    app: redis
  name: redis
spec:
  serviceName: redis
  replicas: 3
  selector:
    matchLabels:
      app: redis
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: redis
    spec:
      initContainers:
        - name: init-redis
          image: redis:6.2.6-bullseye
          env:
            - name: REDIS_PASSWORD
              valueFrom:
                secretKeyRef:
                  key: REDIS_PASSWORD
                  name: redis-secret
          command:
            - bash
            - '-c'
            - |
              set -ex
              # Generate redis server-id from pod ordinal index
              [[ `hostname` =~ -([0-9]+)$ ]] || exit 1
              ordinal=${BASH_REMATCH[1]}
              # Copy appropriate conf.d files from config-map to emptyDir
              if [[ $ordinal -eq 0 ]]; then
                cp /mnt/config-map/master.conf /etc/redis.conf
              else
                cp /mnt/config-map/slave.conf /etc/redis.conf
              fi
              echo "requirepass ${REDIS_PASSWORD)" >> /etc/redis.conf
              echo "masterauth ${REDIS_PASSWORD)" >> /etc/redis.conf
          volumeMounts:
            - name: conf
              mountPath: /etc/
              subPath: redis.conf
            - name: config-map
              mountPath: /mnt/config-map
      containers:
        - image: redis:6.2.6-alpine
          name: redis
          command: [ "redis-server" ]
          args: [ "/etc/redis.conf" ]
          ports:
            - name: redis-port
              containerPort: 6379
          resources: {}
          volumeMounts:
            - name: conf
              mountPath: /etc/
              subPath: redis.conf
            - mountPath: /data
              name: data
      volumes:
        - name: config-map
          configMap:
            name: privatenote-config
  volumeClaimTemplates:
    - metadata:
        name: data
        annotations:
          volume.alpha.kubernetes.io/storage-class: default
      spec:
        accessModes: ["ReadWriteOnce"]
        resources:
          requests:
            storage: 20Mi
    - metadata:
        name: conf
        annotations:
          volume.alpha.kubernetes.io/storage-class: default
      spec:
        spec:
          accessModes: ["ReadWriteOnce"]
          resources:
            requests:
              storage: 10Mi
```

۳) نحوه استفاده از سرویس مستر و رپلیکاها

در statefulset نمی‌توان بصورت همزمان و به هر ترتیب دلخواهی پادها را create یا delete کرد و نمی‌توان پادها را بصورت رندوم آدرس‌دهی کرد و این امر به این دلیل است که پادها identical نیستند، در واقع هر کدام Identity مختص به خود را دارند(این امر باعث تفاوت بین deployment و statefulset شده است).

در واقع یک پاد نقش master را دارد و پادهای دیگر همگی slave هستند و master هم می‌تواند بخواند و هم می‌تواند بنویسد اما دیگر پادها که slave هستند فقط می‌تواند بخوانند زیرا در غیر اینصورت data inconsistency پیش می‌آید.

هر زمانی که master دیتای جدیدی write می‌کند، پادهای slave باید حواسشان باشد که دیتای خود را با master همگام کنند و synchronized باشند، همچنین توجه شود که این کار باعث می‌شود که دیگر دغدغه‌ی همگام بودن داده‌ها را نداشته باشیم(زیرا statefulset خودش این را هندل می‌کند)، همچنین توجه شود که هر پاد volume مجزای خود را داراست درصورتی که همگی از یک تعریف statefulset درست شده‌اند اما در deployment صرفا یک volume برای تمامی پادها وجود دارد.

```
➜  sts kubectl get pods
No resources found in default namespace.
➜  sts kubectl apply -f cm.yaml
configmap/privatenote-config configured
➜  sts kubectl apply -f redis-service.yaml
service/redis created
➜  sts kubectl apply -f redis-sts.yaml
statefulset.apps/redis created
➜  sts kubectl get pods
NAME      READY   STATUS    RESTARTS   AGE
redis-0   1/1     Running   0          3s
redis-1   0/1     Pending   0          0s
➜  sts kubectl get pods
NAME      READY   STATUS    RESTARTS   AGE
redis-0   1/1     Running   0          7s
redis-1   1/1     Running   0          4s
redis-2   0/1     Pending   0          0s
➜  sts kubectl get pods
NAME      READY   STATUS    RESTARTS   AGE
redis-0   1/1     Running   0          14s
redis-1   1/1     Running   0          11s
redis-2   1/1     Running   0          7s
➜  sts kubectl exec -it redis-0 -- redis-cli
Defaulted container "redis" out of: redis, init-redis (init)
127.0.0.1:6379> AUTH verysecretpassword
OK
127.0.0.1:6379> set name Amirmehdy
OK
127.0.0.1:6379> exit
➜  sts kubectl exec -it redis-1 -- redis-cli
Defaulted container "redis" out of: redis, init-redis (init)
127.0.0.1:6379> AUTH verysecretpassword
OK
127.0.0.1:6379> get name
"Amirmehdy"
127.0.0.1:6379> exit
```

همانطور که مشاهده می‌کنیم پادها به درستی ساخته شده‌اند. پاد redis-0 نقش master و بقیه پادها نقش slave را دارند. سپس data synchroznization را چک می‌کنیم تا مطمئن شویم که statefulset موردنظر به درستی کار می‌کند. برای این کار به پاد redis-0 وصل شده‌ایم و مقدار Amirmehdy را در دیتابیس ذخیره کرده‌ایم، سپس به پاد redis-1 متصل شده‌ایم و مشاهده می‌کنیم که این مقدار در این پاد نیز وجود دارد و همگام سازی به درستی صورت گرفته است.

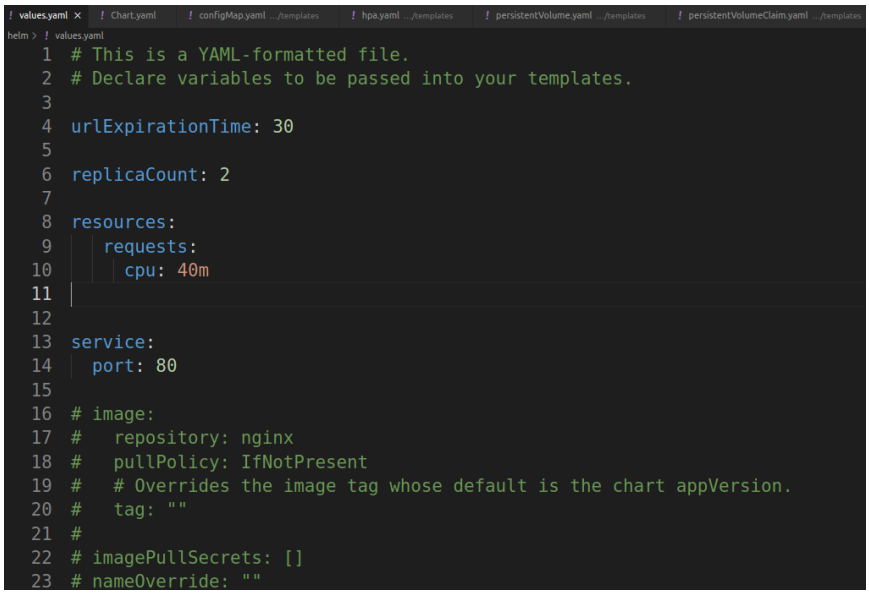
پ) پیاده‌سازی helm chart جهت خودکارسازی

۱) توضیح مختصر ساختار helm chart

Helm، یک package manager برای کوبرنتیز است. در helm، یک چارت، در واقع یک پکیج است که در آن تمامی توصیف‌های لازم برای بالا آوردن یک برنامه و منابع مورد نیاز آن در کوبرنتیز، وجود دارد و با نصب آن چارت، این توصیف‌ها به کوبرنتیز ارسال شده و ایجاد می‌شوند. در شکل زیر، ساختار فایل‌های یک چارت آمده است.

در فایل Chart.yaml، اطلاعاتی کلی در مورد چارت (مانند نسخه و اطلاعات توسعه آن) وجود دارد. فایل values.yaml، برای مشخص کردن مقادیری است که به عنوان پارامتر قابل تغییر، در تولید توصیف منابع مورد نیاز استفاده می‌شود. در پوشه charts، پیش‌نیازهای چارت مورد نظر قرار می‌گیرند. در نهایت، مهم‌ترین بخش یک چارت، فایل‌های template آن هستند. این فایل‌ها، در واقع همان توصیفاتی است که به کوبرنتیز ارسال می‌شوند. با این تفاوت که به جای هارد کد کردن بخش‌های مختلف آن، به فرمت تمپلیت‌های Smarty، داخل {{ }} قرار می‌گیرند. به عنوان مثال، اگر در فایل values.yaml مقدار redisReplicas: 3 را داشته باشیم، در تمپلیت دیپلویمنت ردیس، با نوشتن replicas: {{.Values.redisReplicas}}، مقدار نوشته شده در فایل values.yaml جایگزین می‌شود. همچنین این پارامترها در هنگام نصب چارت و با استفاده از آرگومان‌های CLI قابل مقداردهی هستند.

۲) محتویات و توضیح مختصر پارامترهای تعریف شده در فایل values مربوط به چارت (تعریف درست پارامترها بسیار مهم است)



پارامتر اولی که استفاده کردیم برای زمان منقضی شدن url هاست و پارامتر دوم تعداد replica ها برای اپلیکیشن مورد نظر است در پارامتر بعدی مقدار baseline برای Cpu را مشخص کردیم و در نهایت نیز آخرین پارامتر پورتی که می‌توان با سرویس در ارتباط بود تعیین کردیم.

ج) پیاده‌سازی docker compose جهت خودکارسازی

۱) محتویات و توضیح مختصر docker compose پیاده‌سازی شده

```yaml
version: '3.9'
services:
  privatenote:
    build: .
    image: erfanghasemi/ccnote:1.4
    volumes:
      - ./.env.docker:/env/.env
    ports:
      - 5000:8080

  redis_DB:
    image: redis:6.2.6-alpine
    command: redis-server --requirepass 123456789
    volumes:
      - redis-volume:/data

volumes:
  redis-volume:
```

تا کنون برای داکرایز کردن از docker cli استفاده می‌کردیم. اما با استفاده از docker-compose می‌توان اقدامات مربوط به پروژه را خودکارسازی کرد و دیگر نیاز نباشد برای هر بار دیپلوی کردن و اجرای یک برنامه، دستورات متعدد داکر را وارد کنیم.

ابتدا در خط ۱ ورژنی را که با آن می‌خواهیم docker compose بنویسیم مشخص کرده‌ایم. (ورژن را مشخص می‌کنیم تا مثلا اگر docker compose آپدیت شد و آن را با نسخه‌های جدیدترش اجرا می‌کنیم؛ docker compose نسخه‌ی مورد استفاده این فایل و در واقع نسخه‌ای که این فایل در قالبش نوشته شده‌است را بداند و مشکلی ایجاد نشود)

سپس بخش سرویس‌ها را داریم که در آن همه‌ی سرویس‌هایی که داریم را تعریف می‌کنیم. که در کاربرد فعلی می‌بینیم سرویس‌های مربوط به برنامه(app) و پایگاه‌داده redis را داریم.

پس می‌بینیم در توصیف app از ایمیجی که از dockerfile تولید شده استفاده می‌شود. Volume هم تعریف کرده‌ایم. همچنین portmap انجام داده‌ایم و پورت 5000 را به پورت 8080 داخلی کانتینر پابلیش کرده‌ایم.

در توصیف redis نیز از ایمیج redis:6.2.6-apline استفاده کرده‌ایم و با استفاده از command برایش پسورد 123456789 را قرار داده‌ایم. همچنین برایش volume تعریف کرده‌ایم.

```
➜  Cloud-Computing-Final-Project-main docker-compose up
Starting cloud-computing-final-project-main_redis_1 ... done
Starting cloud-computing-final-project-main_app_1   ... done
Attaching to cloud-computing-final-project-main_redis_1, cloud-computing-final-project-main_app_1
redis_1  | 1:C 01 Jul 2022 19:39:45.865 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
redis_1  | 1:C 01 Jul 2022 19:39:45.865 # Redis version=6.2.6, bits=64, commit=00000000, modified=0, pid=1, just started
redis_1  | 1:C 01 Jul 2022 19:39:45.865 # Configuration loaded
redis_1  | 1:M 01 Jul 2022 19:39:45.867 * monotonic clock: POSIX clock_gettime
redis_1  | 1:M 01 Jul 2022 19:39:45.879 * Running mode=standalone, port=6379.
redis_1  | 1:M 01 Jul 2022 19:39:45.879 # Server initialized
redis_1  | 1:M 01 Jul 2022 19:39:45.880 * Ready to accept connections
app_1    |  * Serving Flask app 'app' (lazy loading)
app_1    |  * Environment: production
app_1    |    WARNING: This is a development server. Do not use it in a production deployment.
app_1    |    Use a production WSGI server instead.
app_1    |  * Debug mode: on
app_1    |  * Running on all addresses (0.0.0.0)
app_1    |    WARNING: This is a development server. Do not use it in a production deployment.
app_1    |  * Running on http://127.0.0.1:8080
app_1    |  * Running on http://172.18.0.3:8080 (Press CTRL+C to quit)
app_1    |  * Restarting with stat
app_1    |  * Debugger is active!
app_1    |  * Debugger PIN: 111-572-777
```