

**Amirkabir University of Technology**  
**(Tehran Polytechnic)**

## گزارش پروژه‌ی نهایی درس رایانش ابری

دانیال حمدی ۹۷۳۱۱۱۱ - امیرحسین رجب‌پور ۹۷۳۱۰۸۵

بهار ۱۴۰۱

## گام دوم

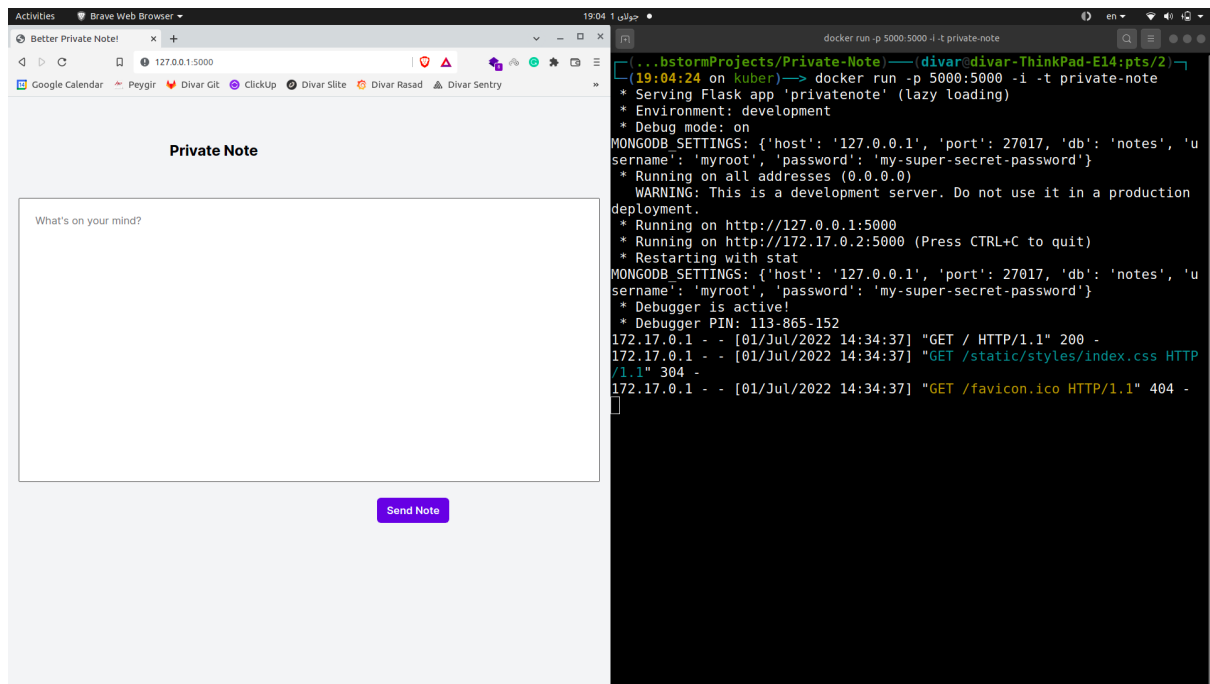
- بیلد کردن image با استفاده از dockerfile ساخته شده:

```
(~/WebstormProjects/Private-Note) (divar@divar-ThinkPad-E14:pts/1)
(18:57:02 on main *)-> docker build -t private-note:latest .
Sending build context to Docker daemon 416.8kB
Step 1/14 : FROM m.docker-registry.ir/python:3.9-slim AS compile-image
3.9-slim: Pulling from python
b85a868b505f: Pull complete
e2be974225ed: Pull complete
339a4e72a1f5: Pull complete
988bab9f4d93: Pull complete
1469e6f7b9e6: Pull complete
Digest: sha256:c01a2db78654c1923da84aa41b029f6162011e3a75db255c24ea16fa2ad563a0
Status: Downloaded newer image for m.docker-registry.ir/python:3.9-slim
--> 32504e766568
Step 2/14 : RUN apt-get update
--> Running in 2dff4538c920
Get:1 http://deb.debian.org/debian bullseye InRelease [116 kB]
Get:2 http://security.debian.org/debian-security bullseye-security InRelease [44.1 kB]
Get:3 http://deb.debian.org/debian bullseye-updates InRelease [39.4 kB]
Get:4 http://security.debian.org/debian-security bullseye-security/main amd64 Packages [161 kB]
Get:5 http://deb.debian.org/debian bullseye/main amd64 Packages [8182 kB]
```

- ارسال ایمج ساخته شده بر روی داکرهاب و نتیجه‌ی آن

```
divar@divar-ThinkPad-E14:~/WebstormProjects/Private-Note | 120x24 | pts/2
(~/WebstormProjects/Private-Note) (divar@divar-ThinkPad-E14:pts/2)
(19:03:18 on kuber)-> docker tag private-note:latest nialda/private-note:latest
(~/WebstormProjects/Private-Note) (divar@divar-ThinkPad-E14:pts/2)
(19:03:23 on kuber)-> docker push nialda/private-note:latest
The push refers to repository [docker.io/nialda/private-note]
0e68bac473ec: Layer already exists
a7f82746f699: Layer already exists
0137f5c9d42b: Layer already exists
d1403b9d2c7b: Layer already exists
4e1acd5ea471: Layer already exists
allc4594e7fb: Layer already exists
1113565264c8: Layer already exists
d442021c7190: Layer already exists
08249ce7456a: Layer already exists
latest: digest: sha256:ff23075be7e2f8d95f60f8257d7e035c93cb9a33ee3c67300b418eb6565e3977 size: 2209
(~/WebstormProjects/Private-Note) (divar@divar-ThinkPad-E14:pts/2)
(19:03:37 on kuber)-> [ ]
```

- تست ایمج بر روی سیستم شخصی (optional)



## • محتویات dockerfile

داکرفایل زیر با روش مولتی استیج نوشته شده است.

```
Dockerfile x
1  >> FROM m.docker-registry.ir/python:3.9-slim AS compile-image
2  RUN apt-get update
3  RUN apt-get install -y --no-install-recommends build-essential gcc
4
5  RUN python -m venv /opt/venv
6  ENV PATH="/opt/venv/bin:$PATH"
7
8  COPY requirements.txt .
9  RUN pip install -r requirements.txt
10
11 COPY src .
12 RUN pip install .
13
14
15 FROM m.docker-registry.ir/python:3.9-slim AS build-image
16 COPY --from=compile-image /opt/venv /opt/venv
17
18 RUN . /opt/venv/bin/activate
19
20 ENV PATH="/opt/venv/bin:$PATH"
21 ENV FLASK_APP='privatenote'
22 ENV FLASK_RUN_HOST=0.0.0.0
23
24 EXPOSE 5000
25
26 CMD ["flask", "run"]
27 |
```

داکتر فایل زیر، بدون روش مولتی استیج ساخته شده است.

```
Dockerfile.tmp x
1  FROM m.docker-registry.ir/python:3.9-slim
2
3  RUN apt-get update -y
4
5  COPY ./requirements.txt /app/requirements.txt
6
7  WORKDIR /app
8
9  RUN pip install -r requirements.txt
10
11 COPY src /app
12
13 EXPOSE 5000
14 ENV FLASK_APP='privatenote'
15 ENV FLASK_ENV='development'
16
17 CMD ["flask", "run", "--host=0.0.0.0"]
```

## گام سوم

- صحت ایجاد منابع بر روی کلاستر

```
divar@divar-ThinkPad-E14: ~/WebstormProjects/Private-Note | 146x38 | pts/2
[~/WebstormProjects/Private-Note]—(divar@divar-ThinkPad-E14:pts/2)
[19:11:24 on kuber]—> kubectl get deployment | grep privatenote
privatenote-deployment 1/1 1 15h
[~/WebstormProjects/Private-Note]—(divar@divar-ThinkPad-E14:pts/2)
[19:11:31 on kuber]—> kubectl get secret | grep mongodb-secret
mongodb-secret Opaque 2 47d
[~/WebstormProjects/Private-Note]—(divar@divar-ThinkPad-E14:pts/2)
[19:11:50 on kuber]—> kubectl get configmap | grep mongodb-configmap
mongodb-configmap 5 17h
[~/WebstormProjects/Private-Note]—(divar@divar-ThinkPad-E14:pts/2)
[19:12:05 on kuber]—> kubectl get service | grep privatenote
privatenote-service ClusterIP 10.233.5.75 <none> 80/TCP 15h
[~/WebstormProjects/Private-Note]—(divar@divar-ThinkPad-E14:pts/2)
[19:12:34 on kuber]—> kubectl get service | grep mongodb
mongodb ClusterIP 10.233.40.23 <none> 27017/TCP 17h
mongodb-service ClusterIP 10.233.19.208 <none> 27017/TCP 15h
[~/WebstormProjects/Private-Note]—(divar@divar-ThinkPad-E14:pts/2)
[19:12:49 on kuber]—> kubectl get deployment | grep mongodb
mongodb 1/1 1 17h
```

- آدرس IP پادها و نحوه ی برقراری ارتباط آن ها و سرویس ساخته شده

```
divar@divar-ThinkPad-E14: ~/WebstormProjects/Private-Note | 146x38 | pts/2
[~/WebstormProjects/Private-Note]
(19:14:18 on kuber) -> kubectl get pod -o wide | grep privatenote
privatenote-deployment-9cdb84f95-btrrv 1/1 Running 0 15h 10.233.108.223 kube7 <none> <none>
(19:14:34 on kuber) -> kubectl get pod -o wide | grep mongodb
mongodb-6fff488d7f-2f5xr 1/1 Running 0 17h 10.233.109.248 kube12 <none> <none>
```

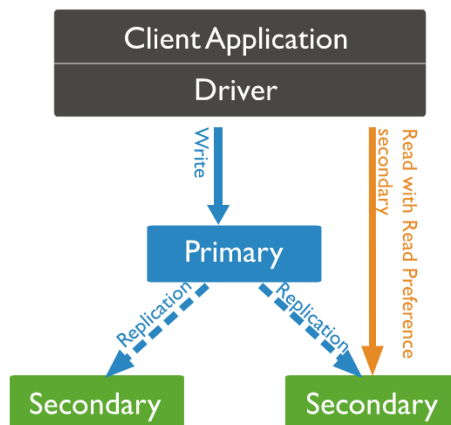
```
divar@divar-ThinkPad-E14: ~/WebstormProjects/Private-Note | 146x38 | pts/2
[~/WebstormProjects/Private-Note]
(19:15:47 on kuber) -> kubectl describe service privatenote
Name: privatenote-service
Namespace: divar-mentorship
Labels: <none>
Annotations: <none>
Selector: app=privatenote
Type: ClusterIP
IP Family Policy: SingleStack
IP Families: IPv4
IP: 10.233.5.75
IPs: 10.233.5.75
Port: <unset> 80/TCP
TargetPort: 5000/TCP
Endpoints: 10.233.108.223:5000
Session Affinity: None
Events: <none>
```

```
divar@divar-ThinkPad-E14: ~/WebstormProjects/Private-Note | 146x38 | pts/2
[~/WebstormProjects/Private-Note]
(19:16:27 on kuber) -> kubectl describe service mongodb-service
Name: mongodb-service
Namespace: divar-mentorship
Labels: <none>
Annotations: <none>
Selector: app=mongodb
Type: ClusterIP
IP Family Policy: SingleStack
IP Families: IPv4
IP: 10.233.19.208
IPs: 10.233.19.208
Port: <unset> 27017/TCP
TargetPort: 27017/TCP
Endpoints: 10.233.109.248:27017
Session Affinity: None
Events: <none>
```

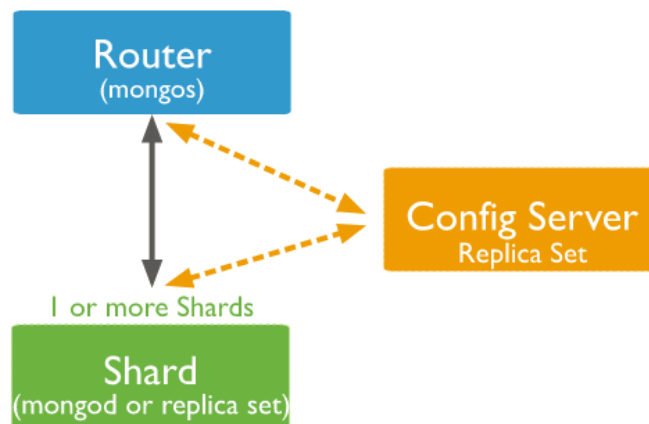
### • دیپلویمنت مربوط به دیتابیس

تعداد پادهای دیپلویمنت دیتابیس مونگو را برابر ۱ قرار دادیم. به طور کلی از مونگودی بی با چند معماری مختلف استفاده می‌شود.

- ۱. Standalone: در این معماری، تنها یک گره از دیتابیس ایجاد می‌کنیم.
- ۲. Replicaset: در این معماری، چند گره از دیتابیس داریم. یک گره، به Primary و دیگر گره‌ها به Secondary تبدیل می‌شوند. تمام درخواست‌های Write به گره Primary فرستاده می‌شوند. درخواست‌های Read اما می‌توانند به گره Secondary ارسال شوند. پس، مرجع داده‌ها گره Primary بوده و گره‌های Secondary باید خودشان را با آن هم‌گام نگه دارند.



۳. Sharded Cluster: در این معماری، چندین خوشه داریم که هر یک با دیگری در ارتباط است.



استفاده از Replication با ایجاد Redundancy (چند نمونه از داده) منجر به Availability بالاتر داده‌ها می‌شود. در این پروژه، برای سادگی، معماری Standalone را برگزیدیم.

## موارد امتیازی

ساختن یک کامپوننت HPA در کلاستر به منظور انجام عملیات Autoscaling

- پارامترهای موجود

پارامترهای Autoscaling می‌تواند external (معیاری خارجی، مستقل از وضعیت منابع کلاستر)، Object (معیاری وابسته به یک شی خاص در کوبرنتیز)، و یا Pods و Resources باشد.

پارامترهای Resource یا همان وابسته به منبع Autoscaling پادها می‌تواند برحسب میزان بهره‌وری و مصرف CPU، یا Disk یا Network Resources یا GPU و Memory باشند.

برای مثال پارامترهای بر اساس مصرف CPU را در زیر توضیح داده‌ایم.

- تعداد هسته‌ی سی‌پی‌یو مصرف شده توسط پاد
  - بهره‌وری سی‌پی‌یو: درصدی از سی‌پی‌یو Node که توسط پاد مصرف شده است.
  - بهره‌وری سی‌پی‌یو: نسبت تعداد هسته‌ی استفاده شده برای پاسخ‌گویی به هر درخواست به میزانی که در کانتینر مشخص شده است.
- لیست کامل این پارامترها در زیر آمده است.

## Disk metrics

Metric Name in the Console	Unit in the Console	Remarks	type	metricName	Default Unit
Disk Write Traffic	KB/s	Pod's disk write rate	Pods	k8s_pod_fs_write_bytes	B/s
Disk Read Traffic	KB/s	Pod's disk read rate	Pods	k8s_pod_fs_read_bytes	B/s
Disk Read IOPS	Times/s	Number of I/O times Pod reads data from disk	Pods	k8s_pod_fs_read_times	Times/s
Disk Write IOPS	Times/s	Number of I/O times Pod writes data to disk	Pods	k8s_pod_fs_write_times	Times/s

## Network

Metric Name in the Console	Unit in the Console	Remarks	type	metricName	Default Unit
Inbound Network Bandwidth	Mbps	Sum of inbound bandwidth of all containers per Pod	Pods	k8s_pod_network_receive_bytes_bw	Bps
Outbound Network Bandwidth	Mbps	Sum of outbound bandwidth of all containers per Pod	Pods	k8s_pod_network_transmit_bytes_bw	Bps
Inbound Network Traffic	KB/s	Sum of inbound traffic of all containers per Pod	Pods	k8s_pod_network_receive_bytes	B/s
Outbound Network Traffic	KB/s	Sum of outbound traffic of all containers per Pod	Pods	k8s_pod_network_transmit_bytes	B/s
Network Packets In	Count/s	Sum of inbound packets of all containers per Pod	Pods	k8s_pod_network_receive_packets	Count/s
Network Packets Out	Count/s	Sum of outbound packets of all containers per Pod	Pods	k8s_pod_network_transmit_packets	Count/s



## Memory

Metric Name in the Console	Unit in the Console	Remarks	type	metricName	Default Unit
Memory usage	MiB	Amount of memory used by the pod	Pods	k8s_pod_mem_usage_bytes	B
Memory Usage (excluding cache)	MiB	Pod Memory Usage, excluding cache	Pods	k8s_pod_mem_no_cache_bytes	B
Memory Utilization (per node)	%	Percentage of total memory of the node used by the Pod	Pods	k8s_pod_rate_mem_usage_node	%
Memory Utilization (per node, excluding cache)	%	Percentage of total memory of the node used by the Pod, excluding cache	Pods	k8s_pod_rate_mem_no_cache_node	%
Memory Utilization (per Request)	%	Percentage of total memory of the Request used by the Pod	Pods	k8s_pod_rate_mem_usage_request	%
Memory Utilization (per Request, excluding cache)	%	Percentage of total memory of the Request used by the Pod, excluding cache	Pods	k8s_pod_rate_mem_no_cache_request	%
Memory Utilization (per Limit)	%	Percentage of Pod memory usage to the Limit value	Pods	k8s_pod_rate_mem_usage_limit	%
Memory Utilization (per Limit, excluding cache)	%	Percentage of Pod memory usage to the Limit value, excluding cache	Pods	k8s_pod_rate_mem_no_cache_limit	%

## GPU

 Note :

The following GPU-related triggering metrics can only be used in EKS clusters.

Metric Name in the Console	Unit in the Console	Remarks	type	metricName	Default Unit
GPU Usage	CUDA Core	Pod GPU usage	Pods	k8s_pod_gpu_used	CUDA Core
GPU Applications	CUDA Core	Pod GPU applications	Pods	k8s_pod_gpu_request	CUDA Core
GPU Utilization (per Request)	%	Percentage of GPU usage to the Request value	Pods	k8s_pod_rate_gpu_used_request	%
GPU Utilization (per node)	%	GPU usage percentage in the node	Pods	k8s_pod_rate_gpu_used_node	%
GPU Memory Usage	Mib	Pod GPU memory usage	Pods	k8s_pod_gpu_memory_used_bytes	B
GPU Memory Applications	MiB	Pod GPU memory applications	Pods	k8s_pod_gpu_memory_request_bytes	B
GPU Memory Utilization (per Request)	%	Percentage of GPU memory usage to the Request value	Pods	k8s_pod_rate_gpu_memory_used_request	%
GPU Memory Utilization (per node)	%	GPU memory usage percentage in the node	Pods	k8s_pod_rate_gpu_memory_used_node	%

- پارامترهای مورد استفاده‌ی ما

نصب metrics-server:

```
kubectl apply -f
```

```
https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
```

برای آزمایش این قابلیت کوبرنتیز، ما بر روی مصرف CPU شرط می‌گذاریم.

- دستور و توصیف مورد استفاده برای ساخت HPA

با دستور

```
kubectl autoscale deployment php-apache --cpu-percent=50 --min=1  
--max=10
```

می‌توانیم یک HPA برای Deployment خود بسازیم. در این دستور پس از مشخص کردن هدف Autoscale که Deployment می‌باشد، نام Deployment را مشخص می‌کنیم. در تصویر زیر، این دستور برای Deployment اپلیکیشن ما اجرا شده است. مشاهده می‌کنیم که آستانه‌ی Autoscaler، بهره‌وری بالای 50% از CPU است. اما اکنون بهره‌وری تنها 3% است. دو ستون بعدی بازه‌ی تعداد مجاز رپلیکاهای دیپلویمنت، و ستون آخر عمر این آبجکت HPA را نشان می‌دهد.

```
divar@divar-ThinkPad-E14:~$ kubectl autoscale deployment privatenote-deployment --cpu-percent=50 --min=1 --max=10
horizontalpodautoscaler.autoscaling/privatenote-deployment autoscaled
divar@divar-ThinkPad-E14:~$ kubectl get hpa | grep priv
privatenote-deployment  Deployment/privatenote-deployment  3%/50%  1  10  1  10s
divar@divar-ThinkPad-E14:~$ kubectl get hpa privatenote-deployment -o yaml
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  annotations:
    autoscaling.alpha.kubernetes.io/conditions: '[{"type":"AbleToScale","status":"True","lastTransitionTime":"2022-07-01T19:41:14Z","reason":"ReadyForNewScale","message":"recommended size matches current size"}, {"type":"ScalingActive","status":"True","lastTransitionTime":"2022-07-01T19:41:14Z","reason":"ValidMetricFound","message":"the HPA was able to successfully calculate a replica count from cpu resource utilization (percentage of request)"}, {"type":"ScalingLimited","status":"False","lastTransitionTime":"2022-07-01T19:41:14Z","reason":"DesiredWithinRange","message":"the desired count is within the acceptable range"}]'
    autoscaling.alpha.kubernetes.io/current-metrics: '[{"type":"Resource","resource":{"name":"cpu","currentAverageUtilization":3,"currentAverageValue":"3m"}]'
  creationTimestamp: "2022-07-01T19:41:08Z"
  name: privatenote-deployment
  namespace: divar-mentorship
  resourceVersion: "3539923823"
  uid: 9185aa0a-ce2c-48ca-8902-9dafc4b6e402
spec:
  maxReplicas: 10
  minReplicas: 1
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: privatenote-deployment
  targetCPUUtilizationPercentage: 50
status:
  currentCPUUtilizationPercentage: 3
  currentReplicas: 1
  desiredReplicas: 1
```

سپس پارامتر هدف Autoscaling را مشخص می‌کنیم، که مثال اسناد آموزشی کوبرنتیز، شرط گذاشتن روی بهره‌وری CPU (در این جا ۵۰٪) بود. سپس بیشینه و کمینه تعداد رپلیکایی که Autoscaler اجازه دارد برای Deployment ما بسازد.

### استفاده از Statefulset به جای Deployment در دیتابیس

- دلیل استفاده از statefulset به جای Deployment

منابع Deployment برخلاف StatefulSet ها (همان‌طور که از اسمشان پیداست) دارای State نیستند. پادهای مختلف از Deployment ساخته و نابود می‌شوند، و Stateشان (مثلاً مجموعه داده‌های سیستمی) ماندگار نبوده و حذف می‌شود.

در مقابل StatefulSet ها از VolumeClaimTemplate ها استفاده می‌کنند تا از حفظ داده‌ها طی ری‌استارت‌ها و یا از بین رفتن پادها اطمینان حاصل کند. این VolumeClaimTemplate برای هر رپلیکای StatefulSet یکتاست.

از پیامدهای Stateful بودن پادهای یک StatefulSet، این است که آن‌ها قابل جایگزینی با هم نبوده، و هر یک ویژگی‌های خاص خودش را دارد. برای مثال، همان‌طور که در قسمت بعد می‌بینیم، برای دیتابیس MongoDB، یک پاد به گره Primary و دیگر گره‌ها به Secondary تبدیل خواهند شد. گره Primary وظیفه نگه داشتن آخرین و به‌روزترین نسخه از داده‌ها را دارد، به همین دلیل درخواست‌های Write را پاسخ می‌دهند. از طرف دیگر گره‌های Secondary درخواست‌های Read را پاسخ می‌دهند تا از لود زیاد روی Primary بکاهند. پس پادهای چنین StatefulSet ی‌یکسان و قابل جایگزینی با هم نخواهند بود.

- توصیف مورد استفاده برای ساخت statefulset

برای Statefulset مورد استفاده در دیتابیس، ابتدا یک سرویس می‌سازیم.

```
apiVersion: v1
kind: Service
metadata:
  name: mongodb-stateful-service
spec:
  selector:
    role: mongo
  ports:
    - protocol: TCP
      port: 27017
```

سپس به توصیف خود Statefulset می‌پردازیم.

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mongodb-stateful
spec:
  selector:
    matchLabels:
      role: mongo
  serviceName: mongodb-stateful-service
  replicas: 3
  template:
    metadata:
      labels:
        role: mongo
        replicaset: MainRepSet
    spec:
      terminationGracePeriodSeconds: 10
      volumes:
        - name: secrets-volume
          secret:
            secretName: shared-bootstrap-data
            defaultMode: 256
      containers:
        - name: mongodb
          image: docker.repos.balad.ir/mongo:4.0.4
          ports:
            - containerPort: 27017
          envFrom:
            - secretRef:
                name: mongodb-secret
            - configMapRef:
                name: mongodb-configmap
          volumeMounts:
            - name: mongodb-persistent-storage
              mountPath: /data/db
      volumeClaimTemplates:
        - metadata:
            name: mongodb-persistent-storage
            annotations:
              volume.beta.kubernetes.io/storage-class: "standard"
          spec:
            accessModes: [ "ReadWriteOnce" ]
            resources:
              requests:
                storage: 1Gi
```

استیت فولست بالا، طبق [راهنمای اسناد آموزشی کوپرنیتیز برای دیتابیس MongoDB](#) ساخته شد.

صحت عملکرد:

پادهای ساخته شده از StatefulSet:

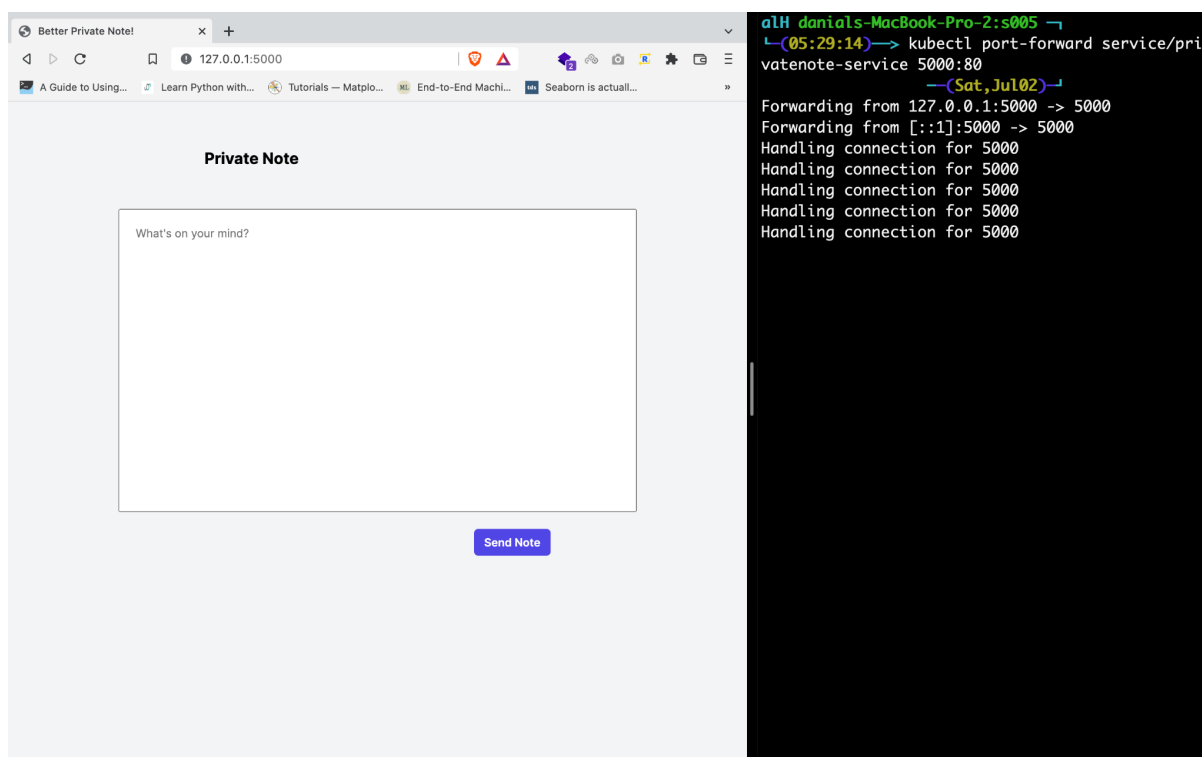
```
(05:28:56 on kuber) → kubectl get pod -w
```

NAME	READY	STATUS	RESTARTS	AGE
mongodb-8445994fcf-9rmvz	1/1	Running	0	8m54s
mongodb-stateful-0	1/1	Running	0	3m36s
mongodb-stateful-1	1/1	Running	0	3m32s
mongodb-stateful-2	1/1	Running	0	3m28s
privatenote-deployment-684ffd8dd6-zmwbh	1/1	Running	0	69s

(Sat, Jul02)

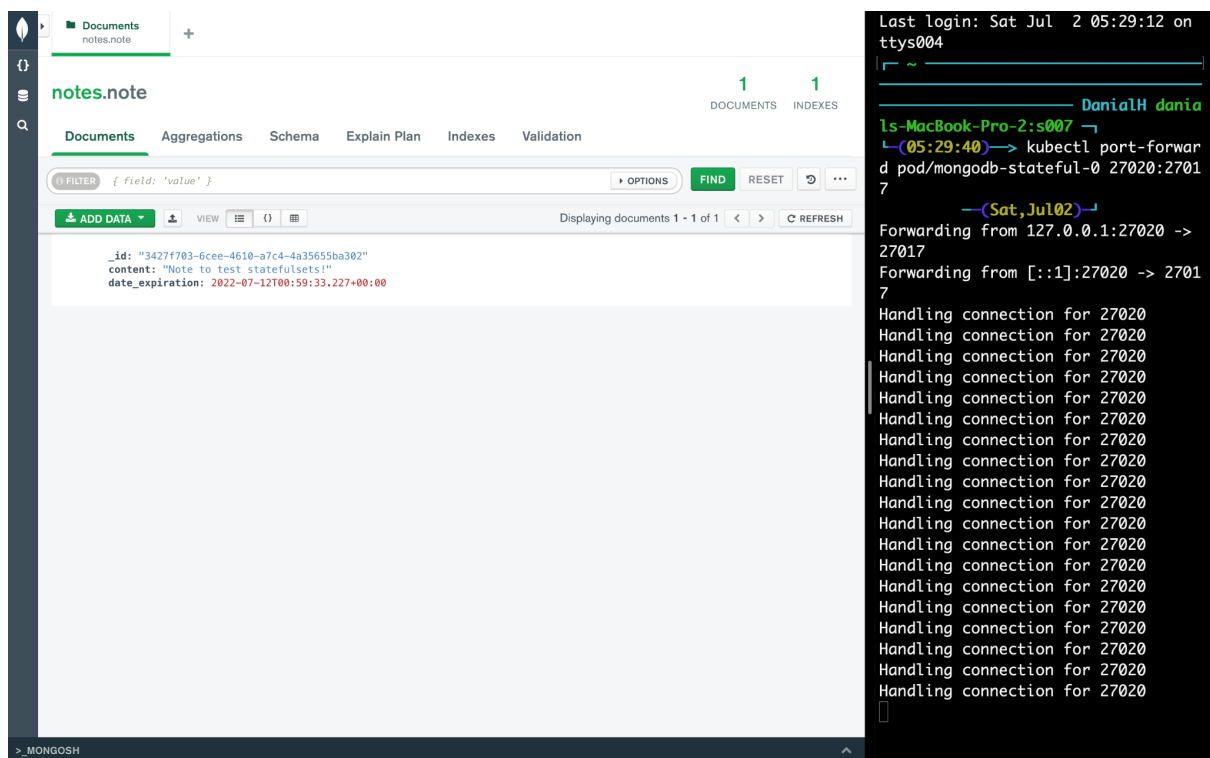
در سرور خود، از متغیر محلی MONGO\_HOST، آدرس دیتابیس Mongo را می‌خوانیم. این مقدار در mongodb-secret تعریف شده بود. این مقدار را به mongodb-stateful-service به‌روزرسانی می‌کنیم، تا به دیتابیس استیت فول متصل شویم.

یک Port-Forward بین پورت ۵۰۰۰ کامپیوتر خود و پورت ۸۰ سرویس اپلیکیشن‌مان می‌سازیم، تا به برنامه‌ی Private Note که توسعه دادیم دسترسی پیدا کنیم.



The image shows a web browser window on the left and a terminal window on the right. The browser window displays the 'Private Note' application, which has a text input field with the placeholder 'What's on your mind?' and a 'Send Note' button. The terminal window shows the command 'kubectl port-forward service/privatenote-service 5000:80' being executed, followed by logs indicating successful port forwarding from 127.0.0.1:5000 to 5000 and handling connections for 5000.

یک Note نوشته و ارسال می‌کنیم. یک Port-Forward دیگر به پاد صفرم (Primary) مونگودی‌بی استیت فول ایجاد می‌کنیم، تا از وجود Note در دیتابیس استیت فول مطمئن شویم.



- نحوه‌ی استفاده از سرویس مستر و رپلیکاها

همان‌طور که گفتیم، سرویس Master سرویسی‌ست که به‌روزترین نسخه‌ی داده را نگه می‌دارد، پس همواره درخواست‌های Write باید به آن نگاشت شود. سرویس‌های Replica باید خود را به داده‌های Master به‌روز نگه داشته، و همچنین به درخواست‌های Read در کنار Master پاسخ دهند. پس هنگام ارسال درخواست‌ها باید به این موضوع توجه کنیم.

### استفاده از Helm Chart به جای Deployment در دیتابیس

- توضیح مختصر ساختار Helm Chart

ساختار فایل‌های Helm Chart به شرح زیر است. فایل Chart.yaml حاوی اطلاعات کلی راجع به چارت ماست. مثلاً نسخه‌ی چارت ما، وابستگی‌هایش به چارت‌های دیگر و... پوشه‌ی templates، شامل تمپلیت‌هایی از توصیفات منابع کوبرنتیز می‌باشد. منابعی مانند Deployment یا Statefulset یا Service، در این پوشه قرار می‌گیرند. اما بعضی مشخصاتشان به صورت متغیری توصیف می‌شود. برای مثال به جای هاردکد کردن تعداد رپلیکاها دیپلویمنت، آن را به صورت {{ replicas.count }} توصیف می‌کنیم. در نهایت در فایل Values.yaml متغیرهایی را که در تمپلیت‌ها مشخص کرده بودیم، مقداری می‌کنیم. مثلاً replicas.count را برابر ۳ می‌گذاریم. فایل Values بسیاری از موقع در واقع پوشه‌ای به نام Values است که به چند فایل (مثلاً متغیرهای Secret و غیر Secret) شکسته می‌شود.

\$ helm create mychart

```
mychart
├── Chart.yaml # Information about your chart, metadata, version and dependency
├── charts # Charts that this chart depends on
├── templates
│   ├── NOTES.txt
│   ├── _helpers.tpl
│   ├── deployment.yaml
│   ├── ingress.yaml
│   ├── service.yaml
│   ├── serviceaccount.yaml
│   └── tests
│       └── test-connection.yaml
└── values.yaml # The default values for your templates
```

- محتویات و توضیح مختصر پارامترهای تعریف شده در فایل Values مربوط به چارت برخی پارامترهایی که مقدارشان را به کمک هلم پویا (Dynamic) کردیم، به شرح زیر است. مقداری مانند تعداد رپلیکا و ایمج تنها کانتینر اپلیکیشن یا دیتابیس مان.
- همچنین مقادیر متغیرهای محیطی تعریف شده در داخل Secret ها را هم به فایل Values هلم منتقل کردیم.
- در نهایت برای نام گذاری Instance های منابع مختلف کوبرنتیز، از Release.Name ی که پکیج فعلی دارد استفاده کردیم. ناگفته پیداست که مقادیر بی شمار پارامتر دیگر را نیز می توانیم به این صورت به فایل Values ببریم.



```
kubernetesClusterDomain: cluster.local

app:
  replicasCount: 3
  image: nialda/private-note:latest

appService:
  port: 80

db:
  image: {{ .Values.docker.repos.balad.ir/mongo:4.0.4 }}
  replicasCount: 1

mongodbSecret:
  mongoInitdbRootPassword: "myroot"
  mongoInitdbRootUsername: "my-super-secret-password"
```

پیاده‌سازی docker compose

- محتویات و توضیح مختصر docker compose پیاده شده

```

version: '3.7'

services:
  private_note:
    build:
      context: .
      dockerfile: Dockerfile.tmp
    environment:
      NOTE_EXPIRATION: "10"
      PORT: "5000"
      MONGO_HOST: private_note_db
      MONGO_PORT: 27017
      MONGO_INITDB_ROOT_PASSWORD: my-super-secret-password
      MONGO_INITDB_ROOT_USERNAME: myroot
      MONGO_INITDB_DATABASE: "notes"
    depends_on:
      - private_note_db
    ports:
      - "5000:5000"

  private_note_db:
    image: docker.repos.balad.ir/mongo:4.0.4
    volumes:
      - private_note_dbdata:/data/db
    environment:
      MONGO_INITDB_ROOT_PASSWORD: my-super-secret-password
      MONGO_INITDB_ROOT_USERNAME: myroot
      MONGO_INITDB_DATABASE: "notes"
    ports:
      - "27017:27017"

volumes:
  private_note_dbdata:

```

در داکر کامپوز خود ۲ سرویس تعریف می‌کنیم. سرویس اول مختص بالا آوردن وب‌اپلیکیشن‌مان (سرور Flask) می‌باشد، و سرویس دوم مختص دیتابیس MongoDB. ایمجی از Mongo که ما استفاده کردیم، در هنگام ساختن کانتینر، بنا به متغیرهای محیطی (Environment Variable) اعم از

- MONGODB\_INITDB\_ROOT\_USERNAME
- MONGODB\_INITDB\_ROOT\_PASSWORD
- MONGODB\_INITDB\_DATABASE

سرویس مونگو را با یوزرنیم و پسورد روت انتخاب شده می‌سازد. در وب‌اپلیکیشن‌مان نیز از همین نام کاربری و پسورد استفاده می‌کنیم (کاربر جدید نساختیم). بنا بر این، در این سرویس هم نیاز به آن متغیرهای محلی داریم. برای سرویس وب‌اپلیکیشن، پورت 5000 کانتینر متناظرش را به پورت 5000 کامپیوتر میزبان نگاشت می‌کنیم. برای سرویس مونگو هم همین کار را برای پورت 27017 انجام می‌دهیم. برای سرویس مونگو نیاز به یک Volume نیز داریم تا از آن برای ذخیره‌سازی داده‌های ایمج استفاده شود.



## آزمون پروژه

