



Minikube

درس : رایانش ابری

استاد مربوطه: دکتر شریفی

ترمه نجارآذری - ۱۴۰۰۴۴۲۱۴۵

بهار ۱۴۰۴

لینک [github:termehNj/minicube](https://github.com/termehNj/minicube)



این گزارش کار روند پیاده‌سازی یک میکروسرویس ساده با استفاده از فریم‌ورک Express در Node.js، کانتینریزه کردن آن با Docker، و در نهایت استقرار آن در Kubernetes را توضیح می‌دهد. همچنین، قابلیت مقیاس‌پذیری و نظارت بر وضعیت سرویس نیز پیاده‌سازی می‌شود.

فاز ۱ طراحی برنامه

در این فاز، ابتدا یک میکروسرویس ساده به عنوان API To-Do List طراحی می‌شود. این API با استفاده از فریم‌ورک Express در Node.js پیاده‌سازی می‌شود و قابلیت‌هایی مانند افزودن، حذف، و نمایش وظایف را دارد.

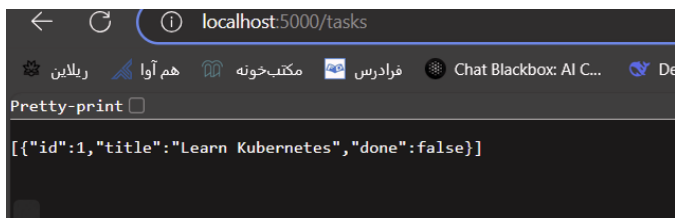
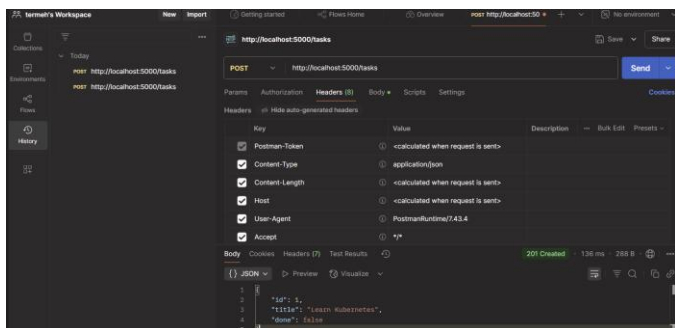
کد API به صورت زیر است:

```
index.js x deployment.yaml service.yaml
index.js > app.post('/tasks') callback
1 const express = require('express');
2 const app = express();
3 const port = 5000;
4
5 app.use(express.json());
6
7 // حافظه موقت برای ذخیره وظایف
8 let tasks = [];
9 let taskIdCounter = 1;
10
11 // لیست تمام وظایف
12 app.get('/tasks', (req, res) => {
13   res.json(tasks);
14 });
15
16 // افزودن وظیفه جدید
17 app.post('/tasks', (req, res) => {
18   const { title } = req.body;
19   const newTask = {
20     id: taskIdCounter++,
21     title: title || "No Title",
22     done: false
23   };
24   tasks.push(newTask);
25   res.status(201).json(newTask);
26 });
27
```

```
index.js x deployment.yaml service.yaml
index.js > app.post('/tasks') callback
28 // دریافت وظیفه بر اساس ID
29 app.get('/tasks/:id', (req, res) => {
30   const taskId = parseInt(req.params.id);
31   const task = tasks.find(t => t.id === taskId);
32   if (task) {
33     res.json(task);
34   } else {
35     res.status(404).json({ error: 'Task not found' });
36   }
37 });
38
39 // حذف وظیفه
40 app.delete('/tasks/:id', (req, res) => {
41   const taskId = parseInt(req.params.id);
42   tasks = tasks.filter(t => t.id !== taskId);
43   res.json({ result: 'Task deleted' });
44 });
45
46 // health check برای readiness/Liveness probe
47 app.get('/health', (req, res) => {
48   res.json({ status: 'OK' });
49 });
50
51 app.listen(port, () => {
52   console.log(`To-Do API running on http://localhost:${port}`);
53 });
54
```

برای تست اپلیکیشن API که به صورت میکروسرویس در حال اجرا است، می‌توانید از Postman استفاده کنید.

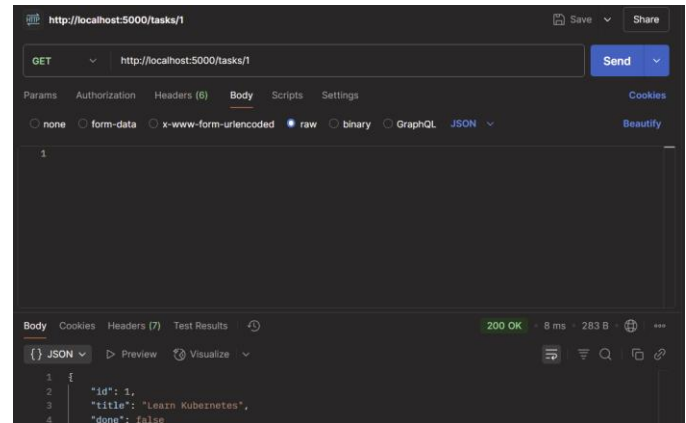
Postman یک ابزار عالی برای تست API ها است و امکان ارسال درخواست‌ها و مشاهده پاسخ‌ها را فراهم می‌کند.



دستورهای ساخت و اجرای Docker :

```
C:\Users\Termeh\Desktop\minikube>docker build -t todo-app .
[*] Building 13.2s (11/11) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 162B
=> [internal] load metadata for docker.io/library/node:18
=> [auth] library/node:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/5] FROM docker.io/library/node:18@sha256:867be01f97d45cb7d89a8ef0b328d3e8267412ebec456441ed8cabcc8c9eed
=> [internal] load build context
=> => transferring context: 2.34MB
=> [2/5] COPY package*.json ./
=> [3/5] COPY package*.json ./
=> [4/5] RUN npm install
=> [5/5] COPY . .
=> exporting to image
=> => exporting layers
=> writing image sha256:72ee28cfef550b192624fd96ebd0cbe23a1fc8d788f69a4eb204b21b23be99
=> naming to docker.io/library/todo-app
```

```
C:\Users\Termeh>docker run -p 5001:5000 todo-app
To-Do API running on http://localhost:5000
|
```



چند تا تسک دیگر هم post کرده و از delete هم به این شکل برای تست استفاده میکنیم:

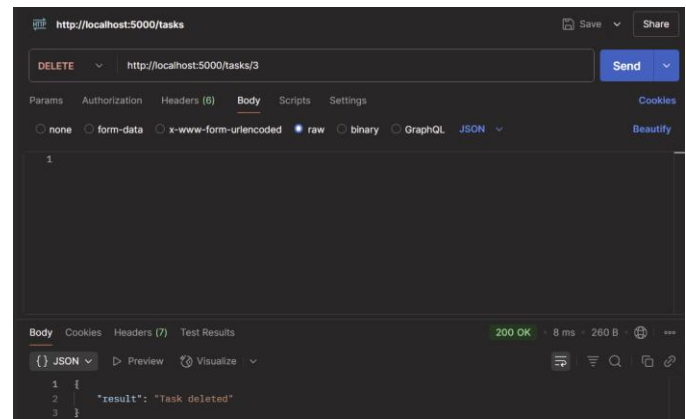
ارسال ایمیج به Docker Hub :

```
C:\Users\Termeh>docker login
Authenticating with existing credentials... [Username: termeh3320]

Info + To login with a different account, run 'docker logout' followed by 'docker login'

Login Succeeded

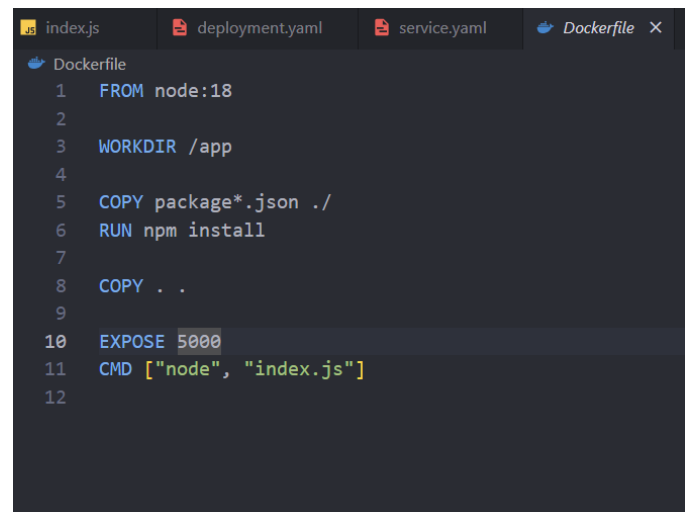
C:\Users\Termeh>docker tag todo-app yourusername/todo-app
C:\Users\Termeh>docker tag todo-app termeh3320/todo-app
C:\Users\Termeh>docker push termeh3320/todo-app
Using default tag: latest
The push refers to repository [docker.io/termeh3320/todo-app]
162ac98321fb: Pushed
3d5bdb5f9978: Pushed
8efd167ebd2a: Pushed
bd95b1bed979: Pushed
8ef4dcd20f76: Mounted from library/node
438e733c9a8c: Mounted from library/node
f7bc5aff5f1: Mounted from library/node
2f3ebdcf6a27: Mounted from library/node
bf9c09fb6f3a: Mounted from library/node
fcb8c0ae5d6: Mounted from library/node
8e3a0e0e61a: Mounted from library/node
2d7ffff7158d: Mounted from library/node
latest: digest: sha256:fe83509f60c7b2b2d665d73cb224d8f0eb617c8829d94275344dff19f74c5f45 size: 2839
```



فاز ۲ کانتینریزه کردن با Docker

در این فاز، اپلیکیشن به یک کانتینر Docker تبدیل می‌شود. برای این کار، ابتدا باید یک Dockerfile ایجاد کرد که نحوه ساخت و راه‌اندازی کانتینر را مشخص کند.

Dockerfile برای اپلیکیشن:



فاز ۳ استقرار در Kubernetes

در این فاز، اپلیکیشن در Kubernetes استقرار می‌یابد. ابتدا Minikube راه‌اندازی می‌شود، سپس فایل‌های YAML برای استقرار برنامه و سرویس ایجاد می‌شود.

راه‌اندازی Minikube :

```
PS C:\Users\Termeh> minikube start
minikube v1.35.0 on Microsoft Windows 11 Enterprise 10.0.26100.3775 Build 26100.3775
- Using the docker driver based on existing profile
- Starting "minikube" primary control-plane node in "minikube" cluster
- Pulling base image v0.0.46 ...
- Updating the running docker "minikube" container ...
- Failing to connect to https://registry.k8s.io/ from inside the minikube container
- To pull new external images, you may need to configure a proxy: https://minikube.sigs.k8s.io/docs/reference/networking/proxy/
- Preparing Kubernetes v1.32.0 on containerd 1.7.24 ...
- Verifying Kubernetes components ...
- Using image gcr.io/k8s-minikube/storage-provisioner:v5
- Enabled addons: storage-provisioner, default-storageclass
- Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
PS C:\Users\Termeh> minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

با این دستور به Kubernetes می گوئید که تعداد پادهای (Pods) مربوط به اپلیکیشن todo-app را به ۳ عدد افزایش دهد. این کار باعث می شود:

- اپلیکیشن شما در ۳ کانیتینر مجزا اجرا شود.
- ترافیک بین این ۳ پاد تقسیم شود. (Load Balancing)
- تحمل خطای سیستم افزایش یابد (اگر یکی از پادها خراب شود، دو تای دیگر باقی می ماند).

HPA

HPA یک ابزار در Kubernetes است که به صورت خودکار تعداد پادهای اپلیکیشن را بر اساس مصرف CPU تنظیم می کند.

- `--cpu-percent=50`: اگر میانگین استفاده از CPU توسط پادها به بیشتر از ۵۰٪ برسد، HPA تصمیم می گیرد که پادهای بیشتری بسازد.
- `--min=2`: حداقل ۲ پاد همیشه فعال باشد.
- `--max=5`: حداکثر می تواند تا ۵ پاد افزایش یابد.

```
PS C:\Users\Termeh> kubectl scale deployment todo-app --replicas=3
deployment.apps/todo-app scaled
PS C:\Users\Termeh> kubectl autoscale deployment todo-app --cpu-percent=50 --min=2 --max=5
horizontalpodautoscaler.autoscaling/todo-app autoscaled
```

افزودن **Liveness Probe** و **Readiness Probe** به فایل `deployment.yaml` یکی از بهترین روش ها برای اطمینان از پایداری و دسترس پذیری سرویس در Kubernetes است.

Liveness Probe - پروب زنده بودن

- هدف: بررسی می کند که آیا اپلیکیشن هنوز زنده و در حال اجراست یا نه.
- اگر شکست بخورد Kubernetes: کانیتینر را ریستارت می کند.
- مناسب برای: زمانی که ممکن است اپلیکیشن دچار مشکل شود ولی همچنان در حال اجرا بماند (مثلاً در حلقه بی پایان گیر کند).

```
index.js deployment.yaml x service.yaml Dockerfile
deployment.yaml > {} spec > {} template
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: todo-app
5  spec:
6    replicas: 2
7    selector:
8      matchLabels:
9        app: todo
10   template:
11     metadata:
12       labels:
13         app: todo
14     spec:
15       containers:
16         - name: todo
17           image: termeh3320/todo-app
18           ports:
19             - containerPort: 5000
20
```

```
index.js x deployment.yaml service.yaml x Dockerfile
service.yaml > {} spec > [ ] ports > {} 0 > # targetPort
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: todo-service
5  spec:
6    selector:
7      app: todo
8    ports:
9      - protocol: TCP
10        port: 80
11        targetPort: 5000
12    type: LoadBalancer # استفاده کند از NodePort برای Minikube
13
```

اعمال فایل های YAML:

```
C:\Users\Termeh\Desktop\miniproj>kubectl apply -f deployment.yaml
deployment.apps/todo-app created

C:\Users\Termeh\Desktop\miniproj>kubectl apply -f service.yaml
service/todo-service created
```

دسترسی به سرویس:

```
PS C:\Users\Termeh> minikube service todo-service
```

NAMESPACE	NAME	TARGET PORT	URL
default	todo-service	80	http://192.168.49.2:31960

فاز ۴ مقیاس پذیری و نظارت

در این فاز، قابلیت مقیاس پذیری و نظارت برای اپلیکیشن فعال می شود.

- هدف: بررسی می‌کند که آیا اپلیکیشن آماده دریافت ترافیک هست یا نه.

- اگر شکست بخورد Kubernetes: آن پاد را موقتاً از load balancing خارج می‌کند ولی آن را ریستارت نمی‌کند.

- مناسب برای: زمانی که اپلیکیشن مدتی طول می‌کشد تا آماده شود (مثلاً اتصال به پایگاه داده).

1. چرا به To-Do API نیاز داریم؟

یک To-Do لیست به کاربران امکان مدیریت وظایف روزانه را می‌دهد. این API پایه برای ساخت اپلیکیشن‌های وب و موبایل مفید است:

- پیاده‌سازی RESTful API ساده
- استفاده در پروژه‌های تمرینی برای یادگیری Express، Docker، Kubernetes
- قابلیت توسعه برای کار با پایگاه‌داده و احراز هویت

افزودن Liveness/Readiness Probe به فایل deployment.yaml:

```
index.js deployment.yaml service.yaml Dockerfile
deployment.yaml > {} spec > {} template > {} spec > [ ] containers > {} 0 > {} readinessProbe
5 spec:
7 selector:
10 template:
11   metadata:
12     labels:
13       app: todo
14   spec:
15     containers:
16       - name: todo
17         image: termeh3320/todo-app
18         ports:
19           - containerPort: 5000
20         livenessProbe:
21           httpGet:
22             path: /health
23             port: 5000
24           initialDelaySeconds: 5
25           periodSeconds: 10
26         readinessProbe:
27           httpGet:
28             path: /health
29             port: 5000
30           initialDelaySeconds: 5
31           periodSeconds: 10
32
```

2. نیازمندی‌ها و اهداف سیستم

اهداف عملکردی:

- افزودن وظایف جدید با POST /tasks
- نمایش لیست وظایف با GET /tasks
- مشاهده جزئیات یک وظیفه با GET /tasks/:id
- حذف یک وظیفه با DELETE /tasks/:id

اهداف غیرعملکردی:

- سریع و ساده برای تست (تاخیر کم‌تر از ۱۰۰ms)
- توسعه‌پذیر برای آینده (قابل اتصال به پایگاه‌داده)
- پشتیبانی از مقیاس‌پذیری با Docker و Kubernetes

3. برآورد ظرفیت و محدودیت‌ها

در حالت فعلی: (In-memory)

- همه وظایف در RAM ذخیره می‌شوند
- با افزایش تعداد وظایف یا کاربر، داده‌ها از بین می‌روند بعد از restart
- مناسب برای محیط‌های تست یا مقیاس کوچک

```
C:\Users\Termeh\Desktop\miniproj>kubectl apply -f deployment.yaml
deployment.apps/todo-app configured

C:\Users\Termeh\Desktop\miniproj>kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
todo-app-5857c8c8f7-gf8mz          0/1     ContainerCreating   0           10s
todo-app-64b7565c7c-l6dtr          0/1     ImagePullBackOff    0           43m
todo-app-d7f4c9979-tvlpk           0/1     ImagePullBackOff    0           43m
```

این فایل‌های Probe به Kubernetes کمک می‌کنند تا وضعیت سرویس را بررسی کند و از بارگذاری درخواست‌ها روی پادهایی که هنوز آماده نیستند جلوگیری کند.

در حالت توسعه یافته:

- **Health Check**: مسیر `/health` همیشه OK 200 برمی گرداند

- ذخیره در دیتابیس واقعی مثل MongoDB یا PostgreSQL
- میلیون ها وظیفه و کاربران پشتیبانی می شود

7. پارتیشن بندی و تکرار داده ها

فعالاً نیازی نیست چون داده ها محلی هستند.

4. طراحی API ها (System APIs)

Method	Endpoint	عملکرد
GET	/tasks	لیست همه وظایف
POST	/tasks	ایجاد وظیفه جدید
GET	/tasks/:id	گرفتن وظیفه بر اساس ID
DELETE	/tasks/:id	حذف وظیفه بر اساس ID
GET	/health	بررسی سلامت سرویس برای Kubernetes probes

8. استفاده از Cache

در حال حاضر داده ها مستقیم در حافظه هستند و نیاز به Cache نیست.

9. لود بالانس

در Kubernetes ، سرویس از طریق `todo-service` در فایل `service.yaml` مدیریت می شود. در Minikube ، با دستور زیر باز می شود:

```
minikube service todo-service
```

اگر `type` برابر `NodePort` باشد، آدرس مثلاً:

```
http://127.0.0.1:32156/
```

خواهد بود.

5. طراحی پایگاه داده (در حال حاضر: حافظه موقت)

در حال حاضر داده ها به صورت زیر در حافظه ذخیره می شوند:

```
let tasks = [
  { id: 1, title: "Buy groceries",
    done: false },
  { id: 2, title: "Finish report",
    done: false }
];
```

برای آینده می توان از:

- **MongoDB** بدون schema
- **PostgreSQL** (با schema دقیق تر) استفاده کرد.

10. پاک سازی داده ها

در حال حاضر حذف با `DELETE /tasks/id` انجام می شود. در نسخه های بعدی:

- تسک هایی که بیش از ۳۰ روز دارند به صورت خودکار حذف شوند
- امکان حذف گروهی یا زمان بندی شده اضافه شود

6. طراحی سیستم و الگوریتم ها

- **ID اتوماتیک**: توسط متغیر `taskIdCounter`
- افزودن تسک: وظیفه جدید به `tasks[]` اضافه می شود
- حذف تسک: با فیلتر کردن `tasks` بر اساس `id`
- دریافت تسک: با `find()` در آرایه

11. امنیت و مجوزها

در حال حاضر:

- هیچ احراز هویتی وجود ندارد
- هیچ Rate Limiting اعمال نشده است

برای نسخه‌های آینده:

- افزودن JWT Auth برای کاربران
 - بررسی داده‌ی ورودی (مثلاً خالی نبودن title)
 - جلوگیری از حملات XSS / Injection
 - محدود کردن تعداد درخواست در هر دقیقه با `express-rate-limit`
-