Minikube

درس : رایانش ابری

استاد مربوطه: دکتر شریفی

ترمه نجارآذری – ۱۴۰۰۴۴۲۱۴۵

بهار ۱۴۰۴

مقدمه

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:

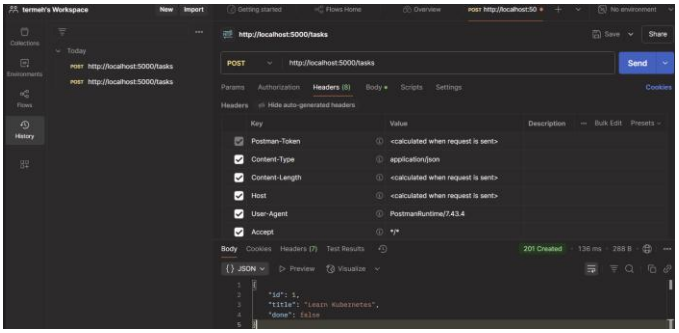
index.js | deployment.yaml | service.yaml

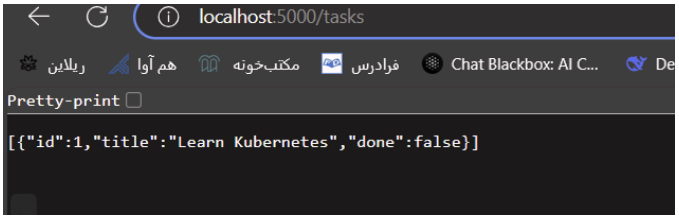
index.js > app.post('/tasks') callback

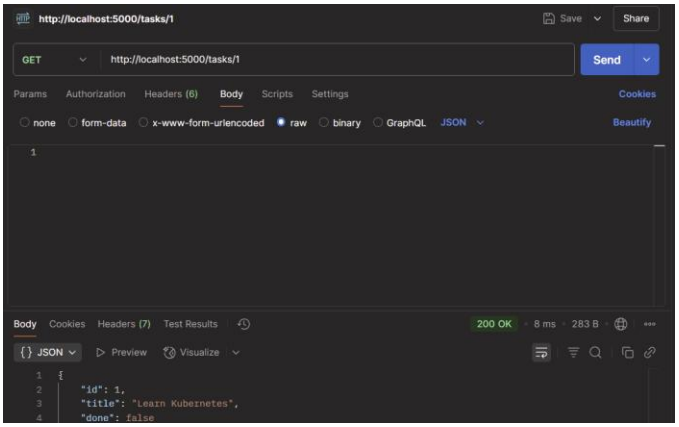
```
const express = require('express');
const app = express();
const port = 5000;

app.use(express.json());

// حافظه موقت برای ذخیره وظایف
let tasks = [];
let taskIdCounter = 1;

// لیست تمام وظایف
app.get('/tasks', (req, res) => {
    res.json(tasks);
});

// افزودن وظیفه جدید
app.post('/tasks', (req, res) => {
    const { title } = req.body;
    const newTask = {
        id: taskIdCounter++,
        title: title || "No Title",
        done: false
    };
    tasks.push(newTask);
    res.status(201).json(newTask);
});

// دریافت وظیفه بر اساس ID
app.get('/tasks/:id', (req, res) => {
    const taskId = parseInt(req.params.id);
    const task = tasks.find(t => t.id === taskId);
    if (task) {
        res.json(task);
    } else {
        res.status(404).json({ error: 'Task not found' });
    }
});

// حذف وظیفه
app.delete('/tasks/:id', (req, res) => {
    const taskId = parseInt(req.params.id);
    tasks = tasks.filter(t => t.id !== taskId);
    res.json({ result: 'Task deleted' });
});

// مسیر health check برای readiness/liveness probe
app.get('/health', (req, res) => {
    res.json({ status: 'OK' });
});

app.listen(port, () => {
    console.log(`To-Do API running on http://localhost:${port}`);
});
```

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

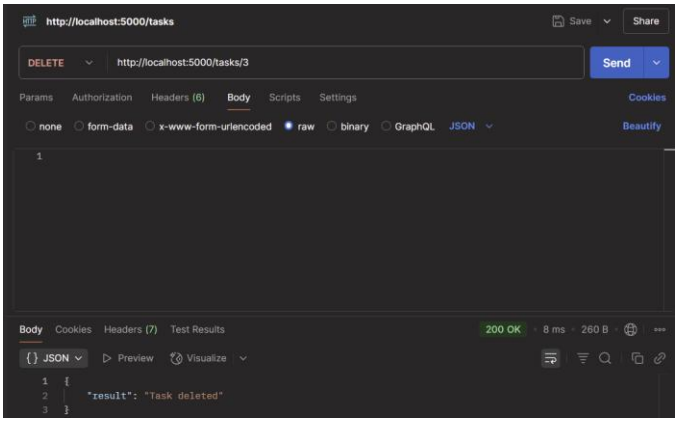
Postman یک ابزار عالی برای تستAPI ها است و امکان ارسال درخواست‌ها و مشاهده پاسخ‌ها را فراهم می‌کند.







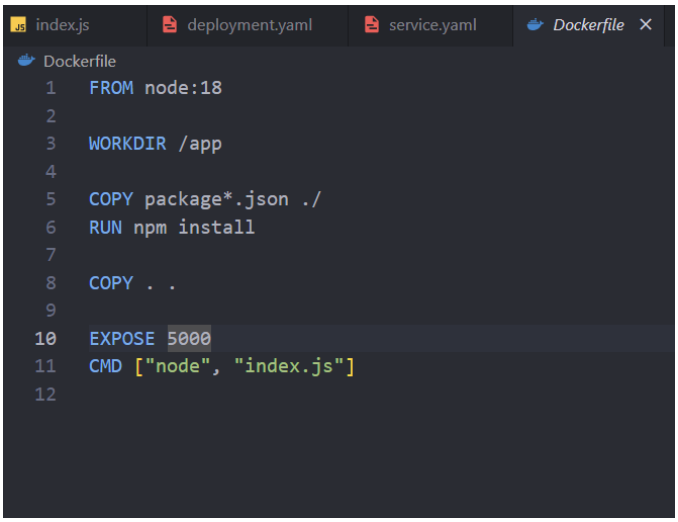
چند تا تسک دیگر هم post کرده و از delete هم به این شکل برای تست استفاده میکنیم:



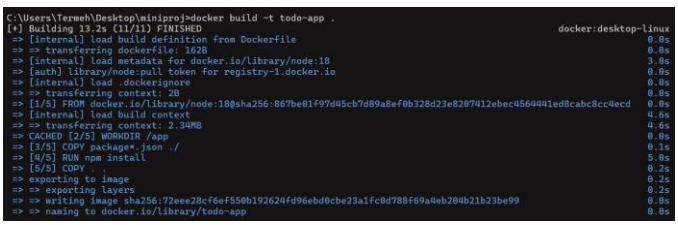
فاز ۲ کانتینریزه کردن باDocker

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:



دستورهای ساخت و اجرای Docker :




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

ارسال ایمیج بهDocker 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
4ef4dcd20f76: Mounted from library/node
438e733c9a0c: Mounted from library/node
f7bcf5aff5f1: Mounted from library/node
2f3ebdcf6a27: Mounted from library/node
bf9c09fb6f3a: Mounted from library/node
fcbb8c0ae5d6: Mounted from library/node
8ce3e08e661a: Mounted from library/node
247fffb7158d: Mounted from library/node
latest: digest: sha256:fe83509f60c7b2b2d665d73cb224d8f0eb617c8829d94275344dff19f74c5f45 size: 2839
```

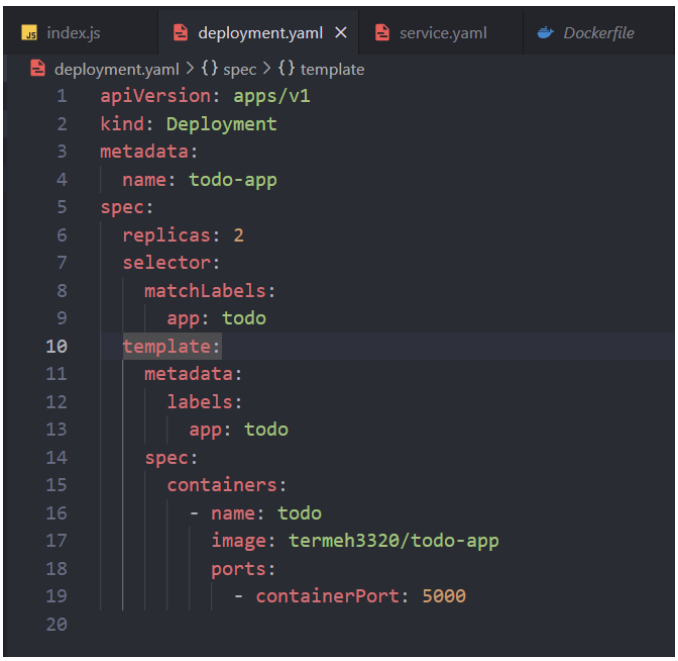
فاز ۳ استقرار درKubernetes

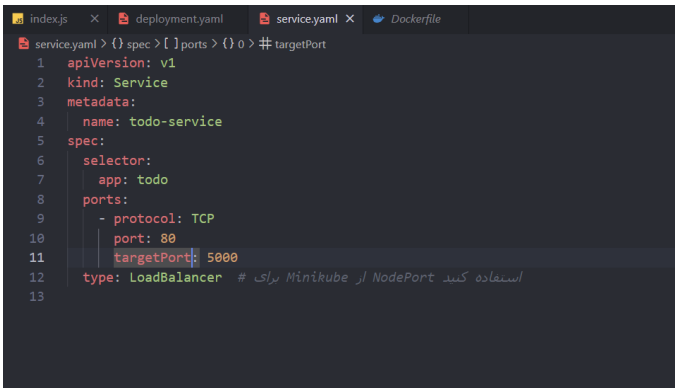
در این فاز، اپلیکیشن در Kubernetes استقرار می‌یابد. ابتدا Minikube راه‌اندازی می‌شود، سپس فایل‌های YAML برای استقرار برنامه و سرویس ایجاد می‌شود.

راه‌اندازیMinikube :

```
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
```

ایجاد فایل‌های YAML برای استقرار (Deployment وService):





اعمال فایل‌هایYAML :

```
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 مقیاس‌پذیری و نظارت

در این فاز، قابلیت مقیاس‌پذیری و نظارت برای اپلیکیشن فعال می‌شود.

Manual Scaling:

با این دستور به Kubernetes می‌گویید که تعداد **پادهای (Pods)** مربوط به اپلیکیشن `todo-app`را به **۳ عدد** افزایش دهد. این کار باعث می‌شود:

- اپلیکیشن شما در ۳ کانتینر 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 می‌مانند).

HPA:

HPA | یک ابزار در Kubernetes است که به‌صورت **خودکار** تعداد پادهای اپلیکیشن را بر اساس مصرف CPU تنظیم می‌کند.

- --cpu-percent=50 : اگر میانگین استفاده از CPU توسط پادها به بیشتر از ۵۰٪ برسد، HPAتصمیم می‌گیرد که پادهای بیشتری بسازد.
- --min=2 : حداقل ۲ پاد همیشه فعال باشد.
- --max=5 :حداکثر می‌تواند 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 روش‌ها برای اطمینان از **پایداری** و **دسترس‌پذیری** سرویس در Kubernetes است.

Liveness Probe - پروب زنده بودن

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

Readiness Probe - پروب آمادگی

- **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

---

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 ۱۰۰(ms
- توسعه‌پذیر برای آینده (قابل اتصال به پایگاه‌داده)
- پشتیبانی از مقیاس‌پذیری با Kubernetes وDocker

---

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 می‌رود بعد از restart
- مناسب برای محیط‌های تست یا مقیاس کوچک

افزودن Liveness/Readiness Probe به فایل deployment.yaml:

```yaml
spec:
  selector:
  template:
    metadata:
      labels:
        app: todo
    spec:
      containers:
        - name: todo
          image: termeh3320/todo-app
          ports:
            - containerPort: 5000
          livenessProbe:
            httpGet:
              path: /health
              port: 5000
            initialDelaySeconds: 5
            periodSeconds: 10
          readinessProbe:
            httpGet:
              path: /health
              port: 5000
            initialDelaySeconds: 5
            periodSeconds: 10
```

```
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-tvlkp    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 توسعه‌یافته:

- ذخیره در دیتابیس واقعی مثل MongoDB یا PostgreSQL
- میلیون‌ها وظیفه و کاربران پشتیبانی می‌شود

4.طراحی API ها(System APIs)

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

---

5. طراحی پایگاه‌داده (در حال حاضر: حافظه موقت)

در حال حاضر داده‌ها به صورت زیر در حافظه ذخیره می‌شوند:

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

برای آینده می‌توان از:

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

---

6. طراحی سیستم و الگوریتم‌ها

- **IDاتوماتیک** :توسط متغیر `taskIdCounter`
- **افزودن تسک** :وظیفه جدید به `tasks[]`اضافه می‌شود
- **حذف تسک** :با فیلتر کردن `tasks`بر اساس `id`
- **دریافت تسک** :با `find()` در آرایه
- **Health Check:** مسیر `/health` همیشه `200 OK` برمی‌گرداند

---

7. پارتیشن‌بندی و تکرار داده‌ها

فعلاً نیازی نیست چون داده‌ها محلی هستند.

8.استفاده از Cache

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

---

9. لود بالانسر

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

```
minikube service todo-service
```

اگر type برابر NodePort باشد، آدرس مثلاً:

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

خواهد بود.

10.پاک‌سازی داده‌ها

در حال حاضر حذف با `DELETE/tasks/id` انجام می‌شود.
در نسخه‌های بعدی:

- تسک‌هایی که بیش از ۳۰ روز دارند به صورت خودکار حذف شوند
- امکان حذف گروهی یا زمان‌بندی شده اضافه شود

---

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 حملاتXSS / Injection
- محدود کردن تعداد درخواست در هر دقیقه با `express-rate-limit`
