بسم الله الرحمن الرحیم

پروژه نهم مهندسی اینترنت

دکتر خامس پناه

علی ممتحن - ۸۱۰۱۰۰۲۱۳

مهدی وجهی - ۸۱۰۱۰۱۵۵۸

ساخت کلاستر با kind

در این مرحله یک کلاستر ساده شامل یک گره مدیر و یک گره کارگر درست می کنیم.

```
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
 - role: control-plane
   image: kindest/node:v1.33.1
 - role: worker
   image: kindest/node:v1.33.1
```

تعریف فرانت اند

تعریف دیپلوی

در دیپلوی فرانت به این صورت عمل کردیم.

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: frontend-deployment
 labels:
   app: frontend
spec:
 replicas: 2
 selector:
   matchLabels:
     app: frontend
 template:
   metadata:
     labels:
       app: frontend
   spec:
     containers:
```

در این فایل پیکربندی، ابتدا با apiVersion: apps/v1 مشخص شده است که از نسخه پایدار و استاندارد API کوبرنتیز برای مدیریت اپلیکیشن‌ها استفاده می‌شود و سپس با kind: Deployment، نوع آبجکت مورد نظر برای ساخت، Deployment تعیین گردیده است تا چرخه حیات اپلیکیشن مدیریت شود. برای این Deployment، در بخش metadata، یک نام منحصر به فرد با عنوان frontend-deployment جهت شناسایی و مدیریت‌های آتی تعریف شده و همچنین برچسب app: frontend جهت سازماندهی بهتر منابع به آن اختصاص یافته است. بخش اصلی تنظیمات در spec قرار دارد که وضعیت مطلوب سیستم را توصیف

می‌کند. در این بخش، دستور replicas: 2 تضمین می‌کند که همواره دو نسخه از اپلیکیشن جهت پایداری سرویس، فعال باقی بماند. برای شناسایی Pod های تحت مدیریت، در بخش selector مشخص شده است که Pod هایی با برچسب app: frontend باید انتخاب شوند. جهت ساخت این Pod ها نیز یک template (الگو) تعریف گردیده که نکته کلیدی در آن، تطابق برچسب موجود در metadata آن با selector است؛ به این ترتیب که در این الگو نیز برچسب app: frontend قرار داده شده تا ارتباط بین Deployment و Pod هایش به درستی برقرار شود. در نهایت، بخش containers در این الگو، محلی است که مشخصات کانتینر برنامه، شامل ایمیج، پورت‌ها و سایر تنظیمات، در آن تعریف خواهد شد.

```
  spec:
    containers:
    - name: frontend
      image: mvajhi/ie-frontend:v1.0
      ports:
      - containerPort: 80
      volumeMounts:
          - name: nginx-config-volume
            mountPath: /etc/nginx/conf.d/default.conf
            subPath: default.conf
    volumes:
      - name: nginx-config-volume
        configMap:
          name: nginx-config
```

این بخش از فایل، مشخصات فنی Pod را در قسمت spec تعریف می‌کند. در بخش containers، یک کانتینر با نام frontend تعریف شده است که ایمیج مورد استفاده برای آن، mvajhi/ie-frontend:v1.0 تعیین گردیده. این کانتینر بر روی پورت داخلی 80 به درخواست‌ها گوش می‌دهد. نکته حائز اهمیت در این تنظیمات، بخش volumeMounts است که به منظور تزریق یک فایل پیکربندی به داخل کانتینر استفاده شده است؛ به این صورت که یک فایل به مسیر /etc/nginx/conf.d/default.conf در داخل کانتینر متصل (mount) می‌شود. گزینه subPath نیز مشخص می‌کند که تنها فایلی با نام default.conf از منبع به این مسیر متصل گردد. منبع این فایل در بخش volumes در سطح Pod تعریف شده است. در این بخش، یک Volume با نام nginx-config-volume ایجاد شده که نوع آن configMap است. این بدان معناست که داده‌های این Volume از یک آبجکت ConfigMap در کوبرنتیز با نام nginx-config خوانده می‌شود. در مجموع، این مکانیزم اجازه می‌دهد که فایل پیکربندی وب‌سرور Nginx به صورت داینامیک و بدون نیاز به ساخت مجدد ایمیج کانتینر، مدیریت و به Pod تزریق شود.

تعریف لود بالانسر

```
apiVersion: v1
kind: Service
```

```
metadata:
 name: frontend
spec:
 type: LoadBalancer
 selector:
   app: frontend
 ports:
   - protocol: TCP
     port: 80
     targetPort: 80
```

یک منبع کوبرنتیز از نوع Service را با استفاده از apiVersion: v1 تعریف می‌کند که هدف اصلی آن، ایجاد یک نقطه دسترسی (endpoint) پایدار و قابل اتکا برای اپلیکیشن frontend است. در بخش metadata، نام این سرویس frontend تعیین شده است که برای شناسایی و ارجاع در داخل کلاستر به کار می‌رود. مهم‌ترین بخش پیکربندی در spec قرار دارد، جایی که type سرویس به صورت LoadBalancer مشخص شده است. این نوع سرویس به کوبرنتیز دستور می‌دهد تا یک Load Balancer خارجی با یک IP آدرس عمومی ایجاد کرده و آن را به این سرویس متصل کند. این سرویس با استفاده از selector مبتنی بر برچسب app: frontend، به صورت خودکار تمام Pod هایی را که این برچسب را دارند (یعنی همان Pod های ساخته شده توسط Deployment قبلی) به عنوان مقصد ترافیک شناسایی می‌کند. در نهایت، در بخش ports، نحوه مسیریابی ترافیک تعریف شده است: ترافیکی که بر روی پروتکل TCP به پورت 80 این Load Balancer می‌رسد، به targetPort یعنی پورت 80 کانتینرهای مقصد ارسال می‌گردد. در نتیجه، این پیکربندی یک راه ارتباطی از اینترنت به اپلیکیشن frontend فراهم کرده و بار ترافیکی را بین تمام نسخه‌های در حال اجرای آن توزیع می‌کند.

تعریف کانفیگ

```
apiVersion: v1
kind: ConfigMap
metadata:
 name: nginx-config
data:
 default.conf: |
   server {
...
```

یک منبع کوبرنتیز از نوع ConfigMap با نام nginx-config را تعریف می‌کند که هدف اصلی آن، جداسازی داده‌های پیکربندی از ایمیج اپلیکیشن است تا امکان مدیریت و به‌روزرسانی تنظیمات بدون نیاز به ساخت مجدد ایمیج فراهم گردد. در بخش data از این ConfigMap، یک کلید با نام default.conf ایجاد شده که مقدار آن، محتوای متنی مربوط به فایل پیکربندی Nginx می‌باشد. این ConfigMap دقیقاً همان منبعی

است که در Deployment مربوط به frontend به آن ارجاع داده شده است؛ در آنجا، یک volume با ارجاع به این ConfigMap تعریف شده و سپس از طریق volumeMount به داخل کانتینر متصل می‌گردد. با استفاده از گزینه subPath: default.conf، مشخص شده است که تنها مقدارِ مربوط به کلید default.conf از این منبع، باید به عنوان یک فایل در مسیر مورد نظر داخل کانتینر قرار گیرد. در نتیجه، این فایل به عنوان یک مخزن مرکزی برای نگهداری تنظیمات Nginx عمل کرده و به Deployment اجازه می‌دهد تا این تنظیمات را به صورت پویا به Pod های در حال اجرا تزریق کند.

تعریف بک اند

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: backend-deployment
 labels:
   app: backend
spec:
 replicas: 1
 selector:
   matchLabels:
     app: backend
 template:
   metadata:
     labels:
       app: backend
   spec:
     containers:
     - name: backend
       image: mvajhi/ie-backend:v1.0
       ports:
       - containerPort: 80
       volumeMounts:
           - name: mysql-root-password-volume
             mountPath: /run/secrets/mysql_root_password
             subPath: password
             readOnly: true
     volumes:
       - name: mysql-root-password-volume
         secret:
# kubectl create secret generic mysql-root-password
--from-file=password=./mysql_root_password.txt
           secretName: mysql-root-password
           items:
             - key: password
               path: password
```

```yaml
---
apiVersion: v1
kind: Service
metadata:
 name: backend
spec:
 selector:
   app: backend
 ports:
   - protocol: TCP
     port: 8080
     targetPort: 8080
```

در این قسمت یک Deployment و یک Service برای کامپوننت backend است. بخش Deployment ساختاری مشابه نمونه frontend دارد و وظیفه مدیریت Pod اپلیکیشن را با برچسب app: backend بر عهده دارد، با این تفاوت که در اینجا تعداد نسخه‌ها (replicas) یک عدد تعیین شده است. تفاوت اصلی و مهم در این Deployment، نحوه مدیریت اطلاعات حساس می‌باشد؛ به جای ConfigMap، از یک Secret با نام mysql-root-password برای تزریق رمز عبور پایگاه داده به اپلیکیشن استفاده شده است. یک volume از این Secret ساخته شده و به صورت فقط خواندنی (readOnly: true) در داخل کانتینر قرار می‌گیرد که این روش، راهکاری امن برای ارائه اطلاعات محرمانه به اپلیکیشن محسوب می‌شود.

بخش دوم فایل، یک Service با نام backend را تعریف می‌کند. تفاوت کلیدی این سرویس با سرویس frontend، در نوع آن است. از آنجایی که type برای آن مشخص نشده، به طور پیش‌فرض از نوع ClusterIP خواهد بود. این یعنی سرویس backend تنها از داخل کلاستر کوبرنتیز قابل دسترسی است و IP خارجی نخواهد داشت. این الگو معمولاً برای ارتباط داخلی بین سرویس‌ها (مانند ارتباط frontend با backend) استفاده می‌شود. این سرویس بر روی پورت 8080 به درخواست‌ها گوش داده و ترافیک را به targetPort یعنی پورت 8080 کانتینر backend هدایت می‌کند.

تعریف پایگاه داده

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: db-data
spec:
 accessModes:
   - ReadWriteOnce
 resources:
   requests:
     storage: 1Gi
---
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
 name: db
spec:
 replicas: 1
 selector:
   matchLabels:
     app: db
 template:
   metadata:
     labels:
       app: db
   spec:
     containers:
       - name: mysql
         image: mysql:5.7
         args:
           - "--lower_case_table_names=1"
         env:
           - name: MYSQL_ROOT_PASSWORD_FILE
             value: /run/secrets/mysql/password
           - name: MYSQL_DATABASE
             value: "shop"
         ports:
           - containerPort: 3306
         volumeMounts:
           - name: db-data
             mountPath: /var/lib/mysql
           - name: mysql-root-password-volume
             mountPath: /run/secrets/mysql
             readOnly: true
     volumes:
       - name: db-data
         persistentVolumeClaim:
           claimName: db-data
       - name: mysql-root-password-volume
         secret:
           secretName: mysql-root-password
           items:
             - key: password
               path: password
---
apiVersion: v1
kind: Service
metadata:
 name: db
spec:
```

```
  selector:
    app: db
  ports:
    - protocol: TCP
      port: 3306
      targetPort: 3306
```

پایگاه داده شامل سه بخش اصلی برای راه‌اندازی پایگاه داده db است: یک PersistentVolumeClaim، یک Deployment و یک Service. بخش جدید و کلیدی در اینجا، PVC با نام db-data است؛ این آبجکت، درخواستی به میزان ۱ گیگابایت برای فضای ذخیره‌سازی پایدار از کلاستر ثبت می‌کند. حالت دسترسی ReadWriteOnce نیز مشخص می‌کند که این فضا تنها توسط یک Node در هر لحظه قابل استفاده است که برای پایگاه‌های داده مناسب می‌باشد.

در بخش Deployment، یک کانتینر mysql:5.7 مدیریت می‌شود. دو نکته مهم در این بخش وجود دارد: اول، اتصال فضای ذخیره‌سازی پایدار که توسط PVC درخواست شده، از طریق volume به مسیر /var/lib/mysql/ در کانتینر است؛ این کار تضمین می‌کند که داده‌های پایگاه داده با از بین رفتن و ایجاد مجدد Pod، حذف نمی‌شوند. دوم، روش امن‌تر برای ارائه رمز عبور است؛ متغیر محیطی MYSQL_ROOT_PASSWORD_FILE به کانتینر می‌گوید که رمز عبور را از یک فایل بخواند و این فایل نیز از طریق volume دیگری که به Secret با نام mysql-root-password متصل است، در کانتینر قرار داده می‌شود. در نهایت، Service تعریف شده از نوع ClusterIP است و یک نقطه دسترسی داخلی و پایدار با نام db بر روی پورت استاندارد مای‌اس‌کیوال یعنی 3306 ایجاد می‌کند تا سایر سرویس‌ها (مانند backend) بتوانند به پایگاه داده متصل شوند.

راه اندازی

برای راه اندازی کلاستر به صورت زیر عمل می کنیم.

```
kind create cluster --config=./cluster-config.yml
```

بعد رمز پایگاه داده رو ذخیره می کنیم.

```
kubectl create secret generic mysql-root-password
--from-file=password=./mysql_root_password.txt
```

حال مانیفست را اعمال می کنیم.

```
kubectl apply -f manifest.yml
```

حال پورت سرویس رو فوروارد می کنیم.

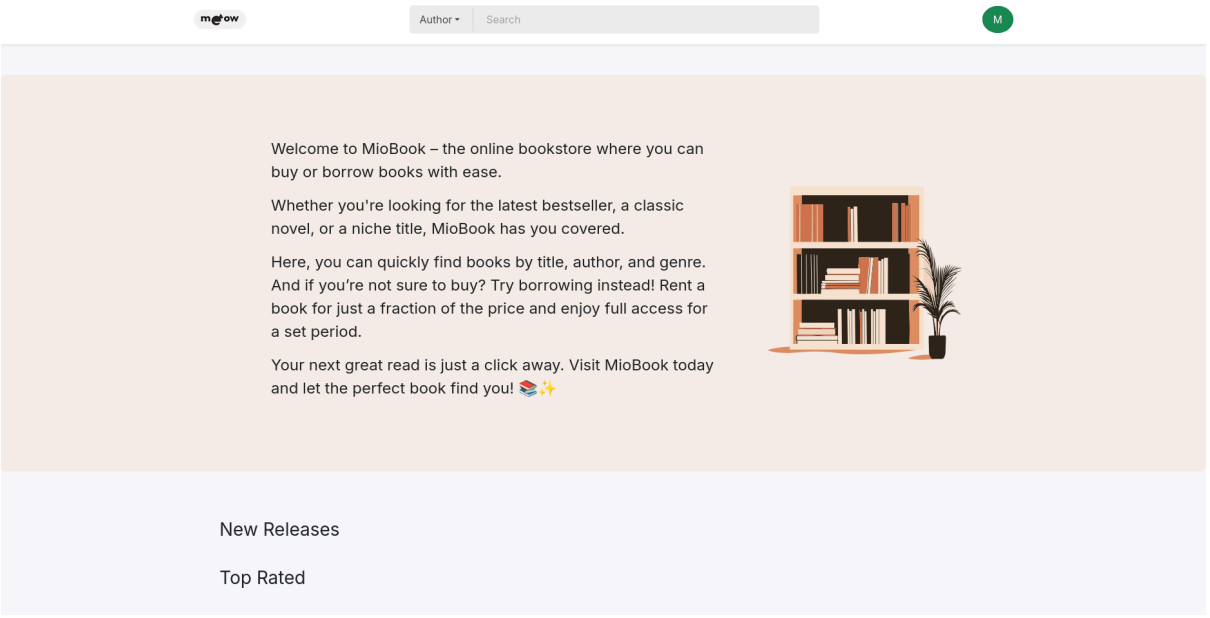
```
kubectl port-forward svc/frontend 8080:80
```

سرویس ها به صورت زیر هست.

```
» k get pod
```

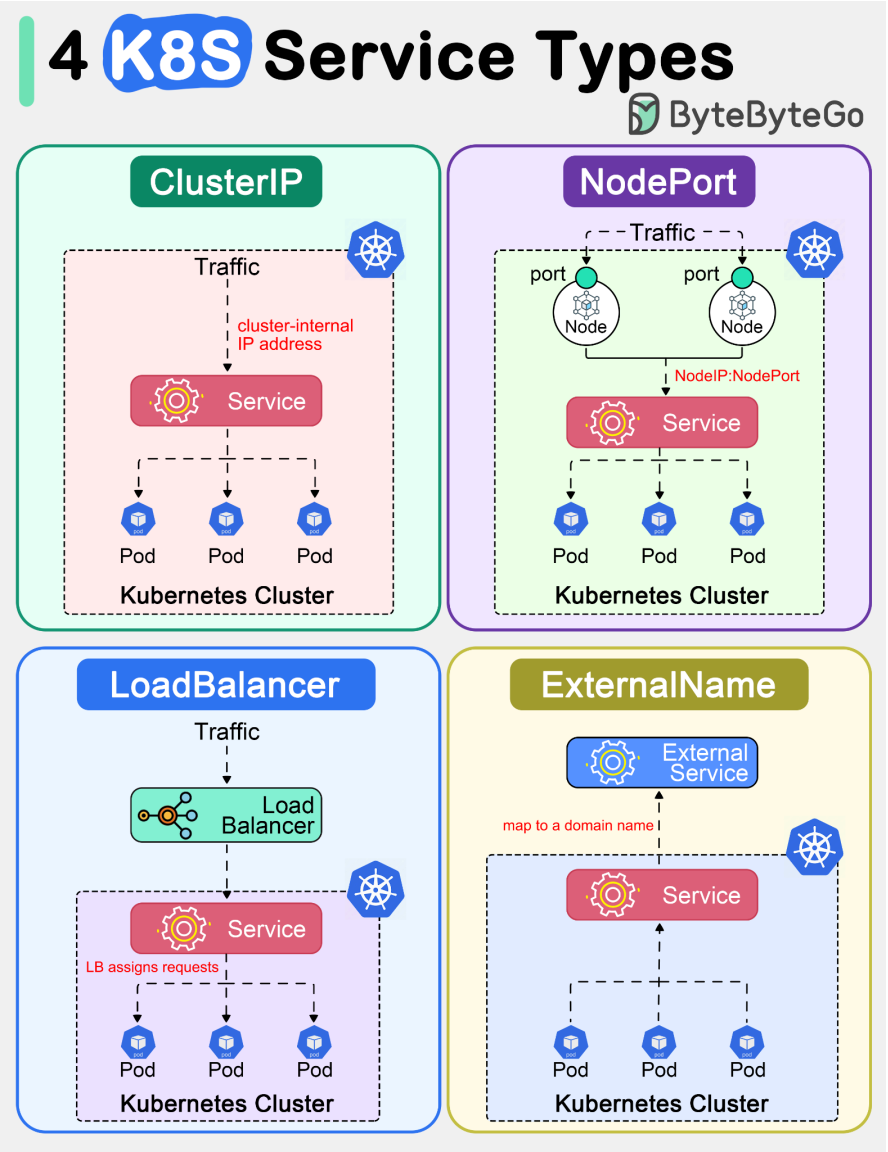
```
NAME                                  READY   STATUS    RESTARTS
AGE
backend-deployment-cfdb94867-gr8s4    1/1     Running   1 (8s ago)
23s
db-86d554479f-mq6xp                   1/1     Running   0
22s
frontend-deployment-6b886476df-jkntr  1/1     Running   0
23s
frontend-deployment-6b886476df-vmtnl  1/1     Running   0
23s
```

در نهایت در مرورگر صفحه سایت بالا میاد و بعد از وارد شدن به صورت زیر است:



سوالات

انواع سرویس ها



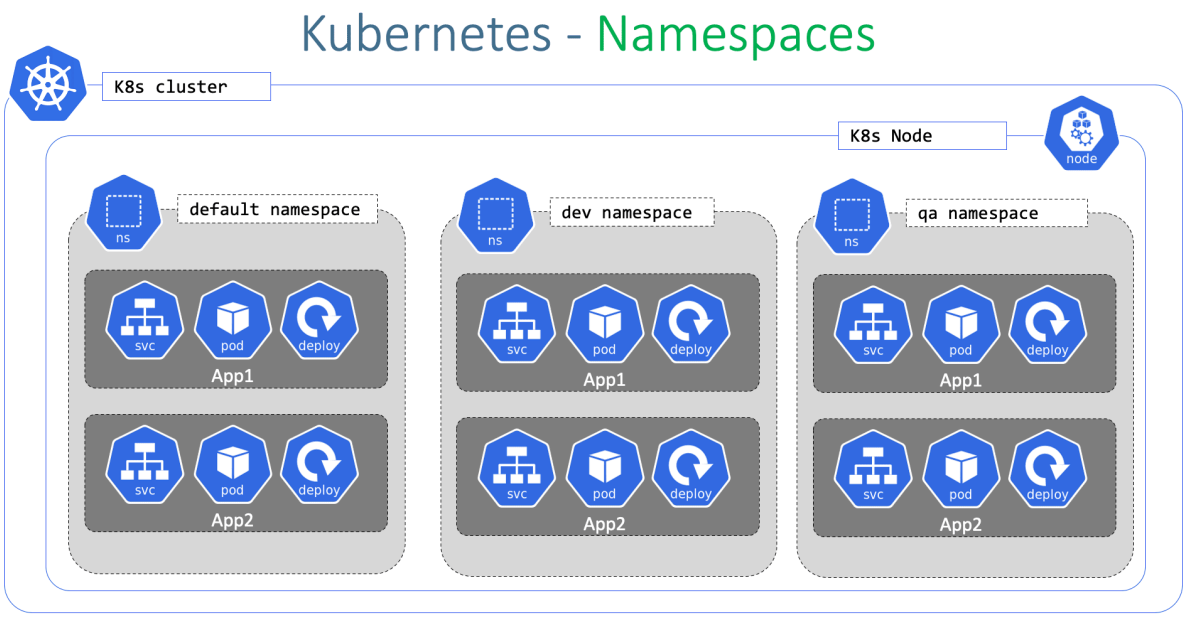
در کوبرنتیز، Service یک راه ثابت برای دسترسی به مجموعه‌ای از Podها (معمولاً یک اپلیکیشن) فراهم می‌کند. انواع اصلی آن عبارتند از:

- ClusterIP: این نوع پیش‌فرض است. یک آدرس IP داخلی در کلاستر به سرویس اختصاص می‌دهد که باعث می‌شود سرویس فقط از درون کلاستر قابل دسترسی باشد. کاربرد اصلی آن برای ارتباط بین سرویس‌های مختلف در کلاستر است (مثلاً ارتباط frontend با backend).
- NodePort: این سرویس، یک پورت ثابت (در محدوده ۳۰۰۰۰-۳۲۷۶۷) را روی تمام Nodeهای کلاستر باز می‌کند. ترافیک ورودی به این پورت روی هر Node، به پورت داخلی سرویس هدایت

می‌شود. کاربرد آن برای تست یا زمانی است که نیاز به دسترسی مستقیم خارجی دارید اما Load Balancer ندارید. محدودیت آن این است که مدیریت پورت‌ها می‌تواند پیچیده باشد.

- LoadBalancer: این نوع، بر اساس NodePort و ClusterIP ساخته می‌شود و از زیرساخت ابری (Cloud Provider) درخواست یک Load Balancer خارجی می‌کند. این Load Balancer یک IP عمومی دریافت کرده و ترافیک را به NodePort سرویس شما هدایت می‌کند. این بهترین راه برای در دسترس قرار دادن سرویس در اینترنت است. محدودیت اصلی آن این است که به پشتیبانی پلتفرم ابری (مانند AWS, GCP, Azure) نیاز دارد و هزینه دارد.
- ExternalName: این نوع، یک رکورد CNAME در DNS داخلی کوبرنتیز ایجاد می‌کند. به جای هدایت ترافیک به Podها، درخواست‌ها را به یک نام دامنه خارجی (مثلاً db.example.com) فوروارد می‌کند.

namespace



Namespace در کوبرنتیز یک کلاستر مجازی برای گروه‌بندی و ایزوله کردن منابع (مانند Pods, Services, Deployments) است. این کار به تیم‌های مختلف اجازه می‌دهد تا بدون تداخل با یکدیگر، در یک کلاستر فیزیکی کار کنند.

تنظیم Namespace پیش‌فرض:

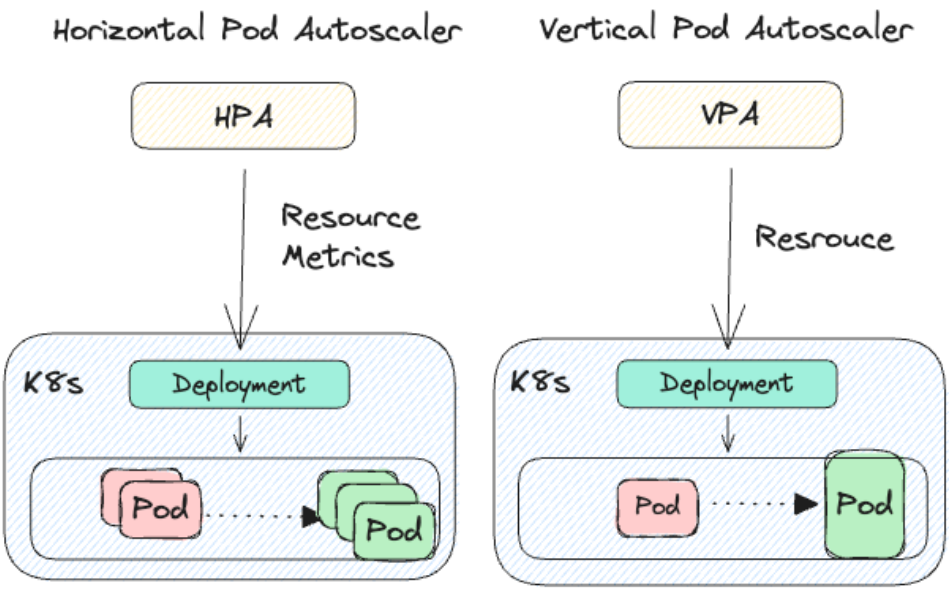
با kubectl: برای تغییر Namespace پیش‌فرض برای context فعلی، از دستور زیر استفاده کنید:

```
kubectl config set-context --current --namespace=your-namespace
```

با ادیت kubeconfig: فایل kubeconfig (معمولاً در مسیر ~/.kube/config) را باز کنید. در بخش contexts، context مورد نظر خود را پیدا کرده و فیلد namespace را به نام Namespace دلخواه تغییر دهید یا اگر وجود ندارد، آن را اضافه کنید.

```
contexts:
- context:
    cluster: my-cluster
    user: my-user
    namespace: new-default-namespace
    name: my-context
```

مقیاس پذیری



در شرایطی که درخواست ها زیاد است باید منابع را افزایش داد برای این کار می توان منابع پاد را زیاد کرد یا تعداد آن را (یا هردو) بسته به نیاز و نوع سرویس می توان یکی از این کار ها را انجام داد. برای این کار منابع را در مانیفست افزایش می دهیم و آن را اعمال می کنیم. می توانیم برای این کار HPA یا VPA تعریف کنیم که اگر مصرف منابع یا متریک خاصی از حدی بیشتر یا کمتر شد منابع و تعداد پاد ها را تغییر دهد. نمونه ای از این تعریف را می توانید مشاهده کنید.

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: nginx-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: nginx-deployment
  minReplicas: 1
  maxReplicas: 5
  metrics:
  - type: Resource
```

```
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 80
```

روند اعمال تغییرات

کوبرنتیز بر اساس یک مدل اعلانی (Declarative) کار می‌کند. وقتی شما یک فایل yaml جدید را با kubectl apply اعمال می‌کنید، در واقع وضعیت مطلوب (Desired State) جدیدی را به سیستم اعلام می‌کنید. واحدهای اصلی که در این پروسه نقش دارند عبارتند از:

1. Controller Manager: این مغز متفکر کوبرنتیز است. وقتی شما یک Deployment را آپدیت می‌کنید، Deployment Controller (که بخشی از Controller Manager است) تفاوت بین وضعیت مطلوب جدید (که در yaml تعریف شده) و وضعیت فعلی سیستم را تشخیص می‌دهد.
2. ReplicaSet: برای اعمال آپدیت به صورت امن، Deployment Controller یک ReplicaSet جدید ایجاد می‌کند و به تدریج تعداد Podهای آن را افزایش می‌دهد، در حالی که تعداد Podهای ReplicaSet قدیمی را کاهش می‌دهد (به این فرآیند Rolling Update می‌گویند).
3. Scheduler: وقتی Podهای جدید توسط ReplicaSet ساخته می‌شوند، Scheduler وظیفه دارد تا بر اساس منابع موجود و محدودیت‌ها، بهترین Node را برای اجرای آن Podها پیدا و تخصیص دهد.

این فرآیند به طور مداوم در یک حلقه تطبیق (Reconciliation Loop) تکرار می‌شود تا وضعیت فعلی سیستم همیشه با وضعیت مطلوبی که شما تعریف کرده‌اید، برابر باشد.

اپراتور

اپراتور (Operator) در کوبرنتیز ابزاری برای خودکارسازی مدیریت نرم‌افزارهای پیچیده است که با ترکیب یک نوع منبع سفارشی (Custom Resource) و یک کنترلر، امکان مدیریت چرخه عمر کامل برنامه‌ها (مثل نصب، پیکربندی، ارتقا، و بازیابی) را فراهم می‌کند. اپراتورها به جای انجام دستی کارها، منطق مدیریتی را به صورت کد پیاده‌سازی می‌کنند و با مشاهده وضعیت منابع، اقدامات لازم را به صورت خودکار انجام می‌دهند؛ برای مثال، یک اپراتور دیتابیس می‌تواند به‌طور خودکار یک پایگاه داده با قابلیت مقیاس‌پذیری، بک‌آپ‌گیری و ریکاوری ایجاد و مدیریت کند.

liveness, readiness و startup

livenessProbe (کاوشگر سلامت): به این سوال پاسخ می‌دهد: "آیا اپلیکیشن در حال اجرا است یا قفل کرده؟". اگر این کاوشگر ناموفق شود، کوبرنتیز نتیجه می‌گیرد که کانتینر دچار مشکل جدی شده و آن را ری‌استارت (restart) می‌کند. این کاوشگر برای بازیابی از بن‌بست‌ها (deadlocks) مفید است.

readinessProbe (کاوشگر آمادگی): به این سوال پاسخ می‌دهد: "آیا اپلیکیشن آماده دریافت ترافیک است؟". ممکن است یک اپلیکیشن در حال اجرا باشد (liveness موفق)، اما به دلیل بارگذاری اولیه یا اتصال به سرویس‌های دیگر، هنوز آماده پاسخگویی به درخواست‌ها نباشد. اگر این کاوشگر ناموفق شود، کوبرنتیز Pod را از لیست Endpoints سرویس حذف می‌کند تا دیگر ترافیکی به آن ارسال نشود. پس از موفقیت مجدد، Pod دوباره به لیست برمی‌گردد.
startupProbe (کاوشگر راه‌اندازی): به این سوال پاسخ می‌دهد: "آیا اپلیکیشن راه‌اندازی اولیه خود را تمام کرده است؟". این کاوشگر برای اپلیکیشن‌هایی که راه‌اندازی آن‌ها زمان‌بر است (مثلاً اپلیکیشن‌های سنگین Java) طراحی شده است. تا زمانی که این کاوشگر موفق نشود، کاوشگرهای liveness و readiness غیرفعال می‌مانند. این کار از ری‌استارت شدن بی‌دلیل اپلیکیشن در حین راه‌اندازی جلوگیری می‌کند.

Deployment, ReplicaSet و DaemonSet

این سه، انواع مختلفی از کنترلرهای مدیریت Pod هستند:

- ReplicaSet: وظیفه اصلی آن این است که تضمین کند تعداد مشخصی از Podهای یکسان همیشه در حال اجرا باشند. این یک کنترلر سطح پایین است و معمولاً به صورت مستقیم از آن استفاده نمی‌شود.
- Deployment: این یک کنترلر سطح بالاتر است که ReplicaSetها را مدیریت می‌کند. Deploymentها قابلیت‌های مهمی مانند آپدیت‌های تدریجی (Rolling Updates) و بازگشت به نسخه قبل (Rollback) را فراهم می‌کنند. شما تقریباً همیشه به جای ReplicaSet از Deployment برای مدیریت اپلیکیشن‌های stateless خود استفاده می‌کنید.
- DaemonSet: این کنترلر تضمین می‌کند که یک کپی از Pod مشخص شده بر روی تمام (یا بخشی از) Nodeهای کلاستر اجرا شود. هر زمان که یک Node جدید به کلاستر اضافه شود، DaemonSet به صورت خودکار آن Pod را روی Node جدید نیز اجرا می‌کند. این کنترلر برای سطح کلاستر مانند جمع‌آوری لاگ (مثل Fluentd)، مانیتورینگ (مثل Prometheus Node Exporter) یا درایورهای ذخیره‌سازی استفاده می‌شود.

requests و limits

requests و limits دو پارامتر کلیدی در بخش resources یک کانتینر هستند که برای مدیریت منابع CPU و حافظه (Memory) به کار می‌روند.

- requests (منابع درخواستی):
  - این مقدار، حداقل منابعی است که کانتینر برای اجرا به آن نیاز دارد و کوبرنتیز تضمین می‌کند که این مقدار را در اختیار آن قرار دهد.
  - Scheduler کوبرنتیز از این مقدار برای تصمیم‌گیری استفاده می‌کند؛ یعنی یک Pod فقط روی Nodeای زمان‌بندی می‌شود که بتواند منابع درخواستی آن را تأمین کند.
- limits (سقف منابع):

- این مقدار، حداکثر منابعی است که یک کانتینر مجاز است از آن استفاده کند.
- اگر کانتینر سعی کند از حافظه بیشتری نسبت به limit خود استفاده کند، توسط سیستم عامل خاتمه می‌یابد (Terminated).
- اگر کانتینر سعی کند از CPU بیشتری نسبت به limit خود استفاده کند، عملکرد آن کاهش می‌یابد (Throttled) اما خاتمه نمی‌یابد.

به طور خلاصه، requests برای تضمین منابع و زمان‌بندی و limits برای جلوگیری از مصرف بیش از حد منابع و تأثیر منفی بر روی سایر اپلیکیشن‌ها استفاده می‌شود.