مبانی رایانش ابری

پروژه پایان ترم

طراحان:

عماد فردوسی، آرمین قربانیان

استاد درس:

دکتر جوادی

مهلت نهایی ارسال پاسخ:

10 تیر ساعت 23:59

مقدمه

در این پروژه، شما یک پلتفرم ساده‌شده KaaS (Kubernetes-as-a-Service) توسعه خواهید داد. این پلتفرم یک API برای استقرار و مدیریت اشیاء Kubernetes، به دو صورت استقرارهای سفارشی‌سازی‌شده و یا از پیش تعریف‌شده ارائه می‌دهد. همچنین به کاربران خدماتی از قبیل HTTP Monitoring برای وب‌سرورهای مستقرشده آن‌ها ارائه می‌دهد.

اهداف این پروژه

در پایان این پروژه، شما:

- اصول Kubernetes و مفاهیم اصلی آن را عمیق‌تر درک خواهید کرد.
- با نحوه ایجاد و مدیریت اشیاء Kubernetes از طریق Kubernetes API آشنا می‌شوید.
- تجربه ساختن یک پلتفرم ساده KaaS را کسب می‌کنید.
- یاد می‌گیرید با استفاده از Helm Charts و Helm برنامه ها را در Kubernetes مستقر کنید.
- با اصول برنامه های Cloud Native و استقرار آنها آشنا می‌شوید.

گام 0) راه اندازی Kubernetes و Helm

برای شروع نیاز به یک کلاستر آماده کوبرنتیز هست. برای این کار و مرور مقدمات کوبرنتیز می‌توانید از دستور العمل‌های موجود در فاز دوم تمرین دوم استفاده کنید.

جهت یادآوری:

نصب Minikube

ابتدا Minikube را نصب کنید. می‌توانید راهنمای نصب رسمی را برای سیستم‌عامل خود دنبال کنید.

شروع Minikube

پس از نصب Minikube، کلاستر Kubernetes خود را با استفاده از فرمان زیر راه‌اندازی کنید:

- minikube start

این فرمان اجزای لازم Kubernetes را دانلود کرده و یک کلاستر تک‌نودی بر روی ماشین محلی شما راه‌اندازی می‌کند.

تأیید نصب Minikube

برای اطمینان از اینکه همه چیز به درستی اجرا شده است، از فرمان زیر استفاده کنید:

- kubectl get nodes

نصب و راه‌اندازی Helm

Helm یک ابزار مدیریتی قدرتمند برای Kubernetes است که به شما امکان می‌دهد تا برنامه‌های Kubernetes و وابستگی‌های آن‌ها را به راحتی نصب، به‌روزرسانی، و مدیریت کنید. Helm به عنوان package manager برای Kubernetes شناخته می‌شود و به شما کمک می‌کند تا از پیچیدگی‌های مدیریت دستی منابع مختلف در Kubernetes بکاهید.

دانلود Helm

ابتدا باینری Helm را دانلود کنید. شما می‌توانید آخرین نسخه را از صفحه انتشارهای GitHub Helm پیدا کنید.

برای مثال، برای دانلود Helm در سیستم‌عامل لینوکس، از فرمان زیر استفاده کنید:

- curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
- chmod 700 get_helm.sh
- ./get_helm.sh

برای macOS، می‌توانید از Homebrew استفاده کنید:

- brew install helm

برای ویندوز، می‌توانید از Chocolatey استفاده کنید:

- choco install kubernetes-helm

یا از راهنمای آن در صفحه رسمی helm استفاده کنید.

برای تأیید اینکه Helm به درستی نصب شده است، فرمان زیر را اجرا کنید:

- helm version

گام 1) راه‌اندازی Ingress Controller

تا کنون شما یک کلاستر Kubernetes راه‌اندازی کرده‌اید و Helm را نیز نصب کرده‌اید. در این مرحله، قصد داریم یک Ingress Controller را برای مدیریت دسترسی خارجی به سرویس‌های در حال اجرا در کلاستر شما نصب کنیم. Ingress Controller به شما این امکان را می‌دهد که ترافیک HTTP و HTTPS را به سرویس‌های مختلف هدایت کنید.

ما از Nginx Ingress Controller که یکی از محبوب‌ترین کنترلرهای ورود برای Kubernetes است، استفاده خواهیم کرد.

1.1 افزودن مخزن Helm برای Nginx Ingress Controller

ابتدا مخزن Helm مربوط به Nginx را اضافه می‌کنیم:

- helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
- helm repo update

1.2 نصب Nginx Ingress Controller

اکنون، با استفاده از Helm آن را نصب می‌کنیم:

- helm install nginx-ingress ingress-nginx/ingress-nginx

برای تأیید نصب، وضعیت پادهای Nginx Ingress Controller را بررسی کنید:

- kubectl get pods --namespace default -l app.kubernetes.io/name=ingress-nginx

برای دسترسی به سرویس از طریق Ingress در محیط محلی، نیاز به تنظیم فایل hosts داریم. آدرس IP ورودی Nginx را پیدا کنید:

- kubectl get svc --namespace default -l app.kubernetes.io/name=ingress-nginx

آدرس IP خارجی را یادداشت کنید و سپس فایل hosts خود را به روز کنید (در لینوکس و macOS: `/etc/hosts` و در ویندوز: `C:\Windows\System32\drivers\etc\hosts`) و خط زیر را اضافه کنید:

- <EXTERNAL-IP> example.local

پس از این می‌توانید سرویس های موجود بر کوبرنتیز که برای آن ها ingress تعریف شده است را با باز کردن مرورگر و وارد کردن آدرس `http://<hostname>.example.local` مشاهده کنید. در قدم های بعد با این مورد روبرو خواهید شد.

گام 2) نوشتن APIهای اصلی KaaS

در این گام، شما باید APIهایی برای پلتفرم خود بنویسید. این APIها شامل عملیات‌های اساسی برای مدیریت برنامه‌ها و منابع Kubernetes خواهند بود. در این گام، سه API اصلی را پیاده‌سازی خواهید کرد:

1. **افزودن اپلیکیشن جدید از کانتینر رجیستری**
2. **گرفتن وضعیت یک دیپلویمنت خاص و پادهای مربوطه**

3. **گرفتن وضعیت همه دیپلویمنت‌ها**

برای پیاده سازی این سرویس‌ها استفاده از کتابخانه های کلاینت کوبرنتیز (به عنوان مثال، client-go برای Go و kubernetes-client برای Python) را برای تعامل با کلاستر Kubernetes خود در برنامه‌ای که می‌نویسید، در نظر بگیرید.

سرویس اول: افزودن اپلیکیشن جدید از Container Registry

برای ایجاد یک اپلیکیشن جدید، نیاز به ساختن اشیاء مختلف Kubernetes دارید. این شامل ایجاد Secrets، Services، Deployments، و Ingress Objects می‌شود. پارامترهای زیر را باید از درخواست دریافت کنید:

- AppName: نام اپلیکیشن
- Replicas: تعداد رپلیکاها
- ImageAddress: آدرس ایمیج در رجیستری کانتینر
- ImageTag: تگ ایمیج
- DomainAddress: آدرس خارجی (در صورت نیاز به دسترسی خارجی، به صورت appName.example.local)
- ServicePort: پورت سرویس
- Resources: شامل CPU، RAM و دیسک
- Envs: متغیرهای محیطی
- Secrets: سیکرت‌های مورد نیاز
- ExternalAccess: قابلیت دسترسی از خارج کلاستر به اپلیکیشن

در صورتی که کاربر برای اپ خود secretهایی را وارد کند، ابتدا می‌بایست شی مربوط به secret را ایجاد کرده و در deployment از آن استفاده کنید.

اگر کاربر نیاز به قابلیت دسترسی خارجی داشت نیز می‌بایست برای این اپلیکیشن یک آبجکت ingress مناسب تعریف کنید.

نمونه ای از درخواست ورودی:

```
{
  "AppName": "MyApp",
  "Replicas": 3,
  "ImageAddress": "dockerhub.com/myapp",
  "ImageTag": "latest",
  "DomainAddress": "myapp.example.com",
  "ServicePort": 8080,
  "Resources": {
    "CPU": "500m",
    "RAM": "1Gi"
  },
  "Envs": [
    {
      "Key": "DATABASE_URL",
      "Value": "postgres://user:password@db.example.com:5432/mydb",
      "IsSecret": true
    },
    {
      "Key": "REDIS_HOST",
      "Value": "redis.example.com",
      "IsSecret": false
    }
  ]
}
```

سرویس دوم: دریافت وضعیت یک اپلیکیشن

در این سرویس می‌بایست وضعیت یک Deployment مشخص و پادهای در حال اجرای آن را برگردانید. برای ورودی باید نام app مورد نظر را دریافت کرده و پس از تعامل با کوبرنتیز یک پاسخ در قالب زیر برگردانید:

- DeploymentName
- Replicas: Total number of desired replicas.
- ReadyReplicas: Number of replicas that are ready.
- PodStatuses: List containing the status of each pod, including:
    - Name
    - Phase (e.g., Running, Pending)
    - HostIP
    - PodIP
    - StartTime

برای پیاده سازی این سرویس نیاز است تا:

- نحوه بازیابی اطلاعات Deployment و پاد را با استفاده از APIهای Kubernetes بدست آورید.
- با نحوه فیلتر کردن و پردازش داده ها برای استخراج فقط اطلاعات مورد نیاز آشنا شوید.
- برای مدیریت مواردی که در آن Deployment مشخص شده وجود ندارد، مدیریت خطا را اجرا کنید.

کتابخانه‌های موجود برای ارتباط با کوبرنتیز تمام قابلیت‌های بالا را دارا هستند. برای استفاده از این قابلیت‌ها به مستندات کتابخانه مورد استفاده رجوع کنید.

نمونه‌ای از پاسخ مورد انتظار:


```
{
  "DeploymentName": "my-app",
  "Replicas": 3,
  "ReadyReplicas": 2,
  "PodStatuses": [
    {
      "Name": "my-app-pod-1",
      "Phase": "Running",
      "HostIP": "192.168.1.101",
      "PodIP": "10.244.1.4",
      "StartTime": "2024-06-07T12:30:00Z"
    },
    {
      "Name": "my-app-pod-2",
      "Phase": "Running",
      "HostIP": "192.168.1.102",
      "PodIP": "10.244.1.5",
      "StartTime": "2024-06-07T12:32:00Z"
    },
    {
      "Name": "my-app-pod-3",
      "Phase": "Pending",
      "HostIP": "",
      "PodIP": "",
      "StartTime": "2024-06-07T12:34:00Z"
    }
  ]
}
```

سرویس سوم: دریافت وضعیت تمام اپلیکیشن ها

این سرویس مشابه API قبلی است، اما به جای یک Deployment وضعیت همه Deploymentها در کلاستر را برمی‌گرداند.
برای پیاده سازی این سرویس از همان روش‌های API قبلی استفاده کنید، اما کوئری را تغییر دهید تا همه Deploymentها را برگرداند.

گام 3) استفاده از Helm برای استقرار

Helm یک package manager برای Kubernetes است که به شما امکان تعریف، نصب و ارتقاء برنامه های پیچیده Kubernetes را می دهد. Helm از یک قالب بسته‌بندی به نام charts استفاده می‌کند. Helm Charts مجموعه‌ای از فایل‌ها است که مجموعه‌ای مرتبط از منابع Kubernetes را توصیف می‌کند.

ابتدا کمی باید با Helm Charts آشنا شوید:

هر نمودار معمولاً شامل:

- Chart.yaml: فایلی که حاوی ابرداده های مربوط به نمودار مانند نسخه، نام و توضیحات است.
- values.yaml: مقادیر پیکربندی نمودار را مشخص می‌کند و به کاربران اجازه می‌دهد تا استقرار را سفارشی کنند.
- templates/: دایرکتوری حاوی فایل های تمپلیت که فایل های مانیفست Kubernetes را بر اساس مقادیر ارائه شده در values.yaml تولید می کند.

با استفاده از دستور زیر می‌توانید یک نمودار جدید برای Helm ایجاد کنید:

- helm create kaas-api

سپس باید:

- Chart.yaml را ویرایش کنید تا توضیحی درباره KaaS API خود اضافه کنید.
- values.yaml را تغییر دهید تا مقادیر پیش‌فرض برای استقرار مانند تعداد نسخه‌ها، مخزن Imageها، image Tag و محدودیت‌های منابع را شامل شود.
- برای مدیریت منابع Kubernetes مانند Services، Deployments و Ingress ، الگوهای موجود در تمپلیت ها را تطبیق دهید. مطمئن شوید که این الگوها از مقادیر values.yaml برای انعطاف‌پذیری استفاده می‌کنند.

یک نمونه از یک تمپلیت:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ .Values.app.name }}
spec:
  replicas: {{ .Values.replicas }}
  selector:
    matchLabels:
      app: {{ .Values.app.name }}
  template:
    metadata:
      labels:
        app: {{ .Values.app.name }}
    spec:
      containers:
      - name: {{ .Values.app.name }}
        image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
        ports:
        - containerPort: {{ .Values.service.port }}
        resources:
          requests:
            cpu: {{ .Values.resources.requests.cpu }}
            memory: {{ .Values.resources.requests.memory }}
          limits:
            cpu: {{ .Values.resources.limits.cpu }}
            memory: {{ .Values.resources.limits.memory }}
```

سپس می توانید با استفاده از دستورات زیر نمودار خود را پکیج کنید:

- helm package kaas-api

و با دستور زیر آن را مستقر کنید:

- helm install kaas-api-release ./kaas-api-0.1.0.tgz

قالب پوشه مربوط به helm شما باید شبیه به قالب زیر باشد:

```
kaas-api/
|
├── Chart.yaml
├── values.yaml
├── charts/
├── templates/
|   ├── deployment.yaml
|   ├── service.yaml
|   ├── ingress.yaml
|   └── NOTES.txt
└── .helmignore
```

گام 4) نوشتن API های self-service

در این مرحله، قابلیت‌های API را گسترش می‌دهید تا شامل استقرار برنامه‌های از پیش تعریف‌شده شود. در اینجا برای سادگی از برنامه‌های از پیش تعریف‌شده به PostgreSQL اکتفا می‌کنیم. این API باید حداقل ورودی کاربر را داشته باشد و در عین حال کانفیگ های لازم برای اپلیکیشن را ایجاد کند. همچنین باید دسترسی خارجی اختیاری را با توجه به تنظیمات کاربر انجام دهد.

نکات مورد توجه:

- این سرویس باید با استفاده از Kubernetes Secrets، اطلاعات کاربری (مانند رمز عبور و نام کاربری) را به صورت خودکار تولید و ذخیره کند.
- از ConfigMaps برای مدیریت تنظیمات PostgreSQL استفاده کنید.
  - نمونه یک کانفیگ postgres:

```
# PostgreSQL configuration file
shared_buffers = 128MB
max_connections = 100
```

- گزینه ای برای کاربران برای درخواست دسترسی خارجی فراهم کنید که بر نحوه پیکربندی سرویس Kubernetes تأثیر می گذارد.

نمونه درخواست ورودی:

```
{
  "AppName": "my-postgres",
  "Resources": {
    "cpu": "500m",
    "memory": "1Gi"
  },
  "External": true
}
```

نکته: در پیاده‌سازی این سرویس باید به تفاوت میان Deployment ها و StatefulSet ها توجه کنید!

گام 5) افزودن جاب HTTP Monitoring و وب‌سرور مربوطه

در این گام قصد داریم وب‌سرورهای کاربران که در کلاستر برای شان بالا آورده‌ایم را مانیتور کنیم و وضعیت سلامت آن‌ها را ذخیره کنیم. برای این کار یک آبجکت کوبرنتیز از نوع CronJob تعریف میکنیم. این جاب به صورت دوره‌ای وضعیت سلامت تمام پادهایی که توسط کاربر برای مانیتور شدن مشخص شده‌اند را پایش می‌کند. به این صورت که برنامه کاربر (در صورت درخواست مانیتور شدن) باید دارای یک HTTP endpoint به صورت زیر باشد:

GET /healthz

که در صورت سلامت status code ای برابر 200 برمی‌گرداند و هر status code دیگری به مثابه عدم موفقیت در نظر گرفته می‌شود. جاب هر 5 ثانیه (برای اینکه برای تغییر این مقدار به redeploy کردن helm chart نیاز نباشد، این مقدار باید کانفیگ‌پذیر باشد و از ConfigMap خوانده شود) یک درخواست HTTP به اندپوینت فوق می‌زند و نتیجه هر بار health check را به صورت صحیح در جدولی مشابه زیر ذخیره یا آپدیت می‌کند:

id	app_name	failure_count	success_count	last_failure	last_success	created_at
1	my-app	2	46	TIMESTAMP	TIMESTAMP	TIMESTAMP

برای تمایز اپ‌هایی که نیاز به مانیتور شدن دارند، باید به هر پادی موقع استقرار در مانیفست آن‌ها یک لیبل به اسم "monitor" اضافه شود که مقدار استرینگی آن رشته "true" یا "false" می‌تواند باشد. (چون در yamlهای کوبرنتیز نمی‌توانیم بولین تعریف کنیم از رشته برای این کار استفاده می‌کنیم). مقدار بولین آن در بدنه درخواست به

endpointهایی که در گام‌های قبل تعریف کردیم باید اضافه شود. همچنین برای تمیز دادن پادهای هر اپ کاربر، باید یک لیبل به اسم "app" در مانیفست پاد مربوطه از طریق API کوبرنتیز اضافه شود که به طور مشابه از بدنه درخواست دیپلوی مقدار آن خوانده می‌شود.

```
selector:
  matchLabels:
    monitor: "true"
    app: "my-app"
```

برای بالا آوردن دیتابیس برای نوشتن و خواندن وضعیت اپ‌ها، از آنجایی که می‌خواهیم توزیع بار بهتر و availability بالاتری داشته باشیم پستگرس را به صورت دو StatefulSet مجزا که یکی مخصوص خواندن (slave) و دیگری مخصوص نوشتن (master) است بالا می‌آوریم. (راهنمایی: برای این کار می‌توانید از ایمیج bitnami/postgresql استفاده کنید و متغیرهای محلی master و slave را مطابقا کانفیگ کنید). پس از تعریف service برای هر کدام از این دو، آدرس سرویس مربوط به نوشتن را از ConfigMap خوانده و در متغیرهای محلی جاب تعریف کنید. اطلاعات حساس از قبیل اسم و رمز عبور دیتابیس باید در secret تعریف شود. برای دیتابیس و جاب، منابع معقولی با استفاده از تعریف limits و requests برای cpu و memory اختصاص دهید. توجه داشته باشید با تعریف volume دیتابیس persist شود.

حال برای رصد وضعیت وب‌سرورهای کاربران، لازم است یک وب‌سرور که به آدرس دیتابیس slave ای که پیشتر تعریف کردیم وصل می‌شود بنویسیم. این وب‌سرور دارای اندپوینت زیر است:

GET /health/{app_name}

که در آن app_name یک path parameter و نشان دهنده نام اپی است که کاربر می‌خواهد تاریخچه وضعیت آن را ببیند. خروجی این اندپوینت به صورت json و شامل همه فیلدهای جدولی است که بالاتر گفتیم.

بخش امتیازی)

- **تعریف HPA**
  - برای وب‌سرور اصلی KaaS یک HorizontalPodAutoscaler تعریف کنید که با انتخاب پارامترهای مناسب آن را auto scale کند.
  - صحت کارکرد آن را در گزارش خود نشان دهید.
- **تعریف probeهای liveness، readiness و startup**
  - از این سه مورد به منظور دنبال کردن وضعیت سلامت پادهای اپلیکیشن استفاده می‌شود. شما باید برای پادهای دیپلویمنت اصلی سیستم که وظیفه مستقر کردن برنامه‌های کاربران را دارد این probe ها را با استدلال مناسب به طرزی تعریف کنید که وظیفه هر کدام به درستی انجام گیرد.

- روش کار و تاثیر هر کدام از این probeها در اپلیکیشن، به علاوه استدلال و منطق پشت پیاده‌سازی خود را در گزارش ذکر کنید.

- **Prometheus**
  - با استفاده از پرومتئوس متریک‌هایی تعریف کنید که حداقل شامل این موارد باشد: تعداد درخواست‌ها، تعداد درخواست‌ها با نتیجه ناموفق، زمان صرف شده در پاسخ به هر درخواست، تعداد خطاهای دیتابیس، مدت زمان پاسخ دیتابیس.
- **Grafana**
  - با استفاده از Grafana داشبوردهایی به منظور مصورسازی بهتر متریک‌های تعریف شده در Prometheus تعریف کنید.

نکات مربوط به تحویل تمرین

- تمرین به‌صورت گروهی در گروه‌های دو نفره انجام می‌شود. برای اطمینان از انجام صحیح و به موقع کار، سریع‌تر نسبت به تشکیل گروه خود اقدام کنید و تقسیم وظایف را به صورت متناسبی انجام دهید.
- تمرین دارای تحویل آنلاین می‌باشد. تسلط هردو عضو گروه بر تمام کدها و مفاهیم الزامی است.
- باید بتوانید صحت عملکرد بخش‌های مختلف سیستم را در ارائه نشان دهید.
- از مراحل انجام دستورات و کامندهای خود اسکرین‌شات تهیه کنید و در یک فایل گزارش کارهای انجام‌شده و دلیل پشت روش پیاده‌سازی توصیه‌شده دستور کار یا روش پیاده‌سازی انتخابی خود را توضیح دهید. همچنین بخش‌های مختلف سیستم را تست کنید و صحت عملکرد آن را در گزارش نشان دهید.
- سوالات خود را می‌توانید با تدریس‌یاران مرتبط مطرح نمایید.
- هرگونه تقلب باعث صفر شدن طرفین می‌شود.

مواردی که باید ارسال شوند:

یک فایل زیپ با نام **StudentID1_StudentID2_CCFinalProject.zip** که وقتی آن را باز می‌کنیم شامل یک پوشه حاوی کدهای شما و یک فایل از گزارش شما می‌باشد.

موفق باشید

تیم تدریسیاری درس مبانی رایانش ابری