模块8作业

作业要求

作业一要求：

> 编写 Kubernetes 部署脚本将 httpserver 部署到 kubernetes 集群，以下是你可以思考的维度
>
> - 优雅启动
> - 优雅终止
> - 资源需求和 QoS 保证
> - 探活
> - 日常运维需求，日志等级
> - 配置和代码分离

作业二要求：

> 除了将 httpServer 应用优雅的运行在 Kubernetes 之上，我们还应该考虑如何将服务发布给对内和对外的调用方。
> 来尝试用 Service, Ingress 将你的服务发布给集群外部的调用方吧
> 在第一部分的基础上提供更加完备的部署 spec，包括（不限于）
>
> - Service
> - Ingress
>
> 可以考虑的细节
>
> - 如何确保整个应用的高可用
> - 如何通过证书保证 httpServer 的通讯安全

本次作业没有严格按照要求来写，没有使用httpserver，而是结合工作中的一个实际服务card-bill（基于java springboot）进行编写，因为目前部门仍然以单体架构为主，本次作业是尝试探讨将现有应用部署在K8S集群中的一个尝试。

服务现状及其改造点

card-bill服务的情况如下：

- 原始启动方式是通过shell脚本启动jar包，因为shell不响应SIG-TERM信号，所以改成直接在POD中通过java -jar的命令启动程序；
- card-bill使用application.properties作为配置文件。配置文件以configmap的形式挂载到相应目录，细节参见“配置和代码分离”小节。
- 密钥配置相关。原服务配置在application.properties中，需要改成secret，详见“编写Secret”小节。
- card-bill的日志输出到日志文件而不是stdout，这里就不改输出的凡是了，将日志使用emptyDir来保存。
- 使用java实现的分布式缓存hazelcast，hazelcast也运行与JAVA主进程中。

作业答案

作业一

> 编写 Kubernetes 部署脚本将 httpserver 部署到 kubernetes 集群，以下是你可以思考的维度
>
> - 优雅启动
> - 优雅终止
> - 资源需求和 QoS 保证
> - 探活
> - 日常运维需求，日志等级
> - 配置和代码分离

编写Dockerfile

```
FROM java:8
LABEL seg=card
COPY ./cardbill-1.0.jar /app/
EXPOSE 9090
WORKDIR /app
ENTRYPOINT nohup java
CMD -server -Xmx1G -Xms1G \
  -XX:+UseG1GC -XX:MaxGCPauseMillis=20 \
  -Djava.security.egd=file:/dev/./urandom \
  -XX:InitiatingHeapOccupancyPercent=35 -XX:+DisableExplicitGC -
Djava.awt.headless=true \
  -jar cardbill-1.0.jar  \
  > /dev/null 2>&1 &
```

构建镜像

```
sudo docker build -t dxktt/cardbill:latest .
sudo docker login
sudo docker push dxktt/cardbill
```

编写configmap

该java应用使用properties文件，文件名为application.properties，可以通过如下命令创建configmap

```
kubectl create configmap card-config --from-file=application.properties
```

编写Secret

主要保存原始card-bill服务application.properties文件中的密钥，因为secret对象无法以yaml活properties文件的形式挂载到POD中，所以决定用环境变量的形式进行传递。但原始card-bill服务的配置项是类似`card.bill.bond.privatekey`的形式，命名不符合shell环境变量的要求，还好spring支持灵活的解析，故将其名字改为`cardBillBondPrivatekey`的形式。

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: card-secret
type: Opaque
data:
  cardBillBondPrivatekey: 
TUlJQ2RRSUJBREFOQmdrcWhraUc5dzBCQVFFRkFBU0NBbDh3Z2dKYkFnRUFBb0dCQUpJY1d6R09LM1Bu
MkplQ1Q2aUxaTmcyTEpYZFpSNDNDU1pCUlFzeXkrenVCVjNBaXpOUEs4eStPTkZkalA4Vnd1SGtlWnFy
YWU3Nno3cEgvbS9CK2crUGxRSi90STNRdjc3Uytwcjl5cS9GdkhOWFBySzZKNjBTWXNRUWlVRHJFWTI1
QTNSMUpWWHl0akFhRXdjVFcvbld1RUtuWTZ4SHNNdWsyS3RyRkNiTkFnTUJBQUVDZ1lCQisOUkRVTXpX
dnZEVkd3Z1dPVDBGVnhHemJCUHFIVGlJaWhsQUFXU09UcjhsbzNvNVpadUdEckZoK3psZFNOcUZZTVI1
ME1jNklSSUdNZFQ1YTBRZGE5TzI4YWgxWE5EN2J0MFJ3eXRGWXJGd1JsSUxnbzVrREVBc0drMlpwd0Na
RkJ4RGRUTjNPalgxaWM3YlViMkVhM3VuUjdDOGUySXVUdDIyQjYyMmtRSkJBTTQ0Y3FQRG9PejI1OVdL
b28rOU81YTB3THEwWEZIcjZKR29lbEZ5VFp4VkFqRkQyOTIwdlpoZnNZdWNmaGVSZFVMRzhxamNHeElo
M2RSK290bVB0T3NDUVFDMVlXSEMwOFJDVzlDNmxsWHFRVWRjVktmSXRCNGI1QnJvc2xQQ1FoQnZXTXR0
MG12Ni9jR21ZSFlNRGhIMDk4RktmTnl1T1htbFpReVpJWTlKTlFVbkFrQmlmaGxhMkFPVFNmVytTdUdM
YjJEM2dNc1FJVFo2dGdhMUVOZWc4TXZhbjh0VlJrakxvaGh2bnp6SUptMGV5bzFMNElOalhRVGM2eW5z
NUxMWTRFY1RBa0F0YXBicUM5Q0dOU3JNcmJ3UmpncnFTYUJCbHlnNjM2RUZSekhGNEhqeENKRFhqT3JR
ekIxakphQXBZSlNLZkViLytjZmdNREFQZy8vbGtBdXF6d1hkQWtCZkxuUWVtTHY3UWFMc0Rwcm9IVXhh
Unh2RWJqOWNpN1FzcnkyOXJwTFFZRnR2b1dhY1hpSFRqZmYyZExYR1ZUQ3N1MGZseHNDcW9hV3BsVFhP
VVFDYwo=
  cardBillBondPublickey: 
TUlHZk1BMEdDU3FHU0liM0RRRUJBUVVBQTRHTkFEQ0JpUUtCZ1FDU0hGc3hqaXR6NTlpWGdrK29pMlRZ
Tml5VjNXVWVOd2ttUVVVTE1zdnM3Z1Zkd0lzelR5dk12ampSWFl6L0ZjTGg1SG1hcTJudStzKzZSLzV2
d2ZvUGo1VUNmN1NOMEwrKzB2cWEvY3F2eGJ4elZ6Nnl1aWV0RW1MRUVJbEE2eEdOdVFOMGRTVlY4cll3
R2hNSEUxdjUxcmhDcDJPc1I3RExwTmlyYXhRbXpRSURBUUFCCg==
  cardBillBondAesSaltKey: SlhPOHRUVTdJZwo=
  cardBillBondAeskey: WXRSTnUwZUQxZExRci91L3BiNkFNdz09Cg==
  cardBillBondVectorKey: U3puMjFFSTJLbEpadjlXWgo=

```

注意：因为密钥本来就是用base64编码保存的，为了防止base64解码后出现null字符导致OCI报错，需要做两次BASE64编码！！

编写Deployment

配置和代码相分离

springboot本身就支持配置和代码分离，虽然jar包中本身就包含一个默认的application.properties文件，但只要在jar包所在目录的config目录下也有一个application.properties文件，就会覆盖jar包中原有的配置文件。所以考虑将application.properties做成一个configmap挂载到${HOME}/config目录下。

> ${HOME}是card-bill服务的运行目录，在Dockerfile中被定义为POD中的`/app`目录。

configmap的详情参见“编写configmap”小节。

```
          volumeMounts:
          - mountPath: /app/config   # spring boot应用优先读取jar包所在目录的config子
目录下的配置
            name: config-volume
            readOnly: true
...
      volumes:
      - name: config-volume
        configMap:
          name: card-config
      - name: log-volume
        emptyDir: {}                # 使用emptyDir作为日志目录
```

独立保存密钥

存放到secret中，以环境变量的形式传递给新的card-bill服务，参见“编写Secret”小节。

```
          env:
            - name: cardBillBondPrivatekey
              valueFrom:
                secretKeyRef:
                  name: card-secret
                  key: cardBillBondPrivatekey
            - name: cardBillBondPublickey
              valueFrom:
                secretKeyRef:
                  name: card-secret
                  key: cardBillBondPublickey
            - name: cardBillBondAesSaltKey
              valueFrom:
                secretKeyRef:
                  name: card-secret
                  key: cardBillBondAesSaltKey
            - name: cardBillBondAeskey
              valueFrom:
                secretKeyRef:
                  name: card-secret
                  key: cardBillBondAeskey
            - name: cardBillBondVectorKey
              valueFrom:
                secretKeyRef:
                  name: card-secret
                  key: cardBillBondVectorKey
```

以为密钥BASE64解码后包含null字符，导致`OCI runtime create failed`，所以这样做还有问题，需要改造

日志

使用emptyDir卷config-volume保存日志。

```
          volumeMounts:
          - mountPath: /app/config   # spring boot应用优先读取jar包所在目录的config子目录下的配置
            name: config-volume
            readOnly: true
          - mountPath: /app/logs     # 日志目录
            name: log-volume
...
      volumes:
      - name: config-volume
        configMap:
          name: card-config
      - name: log-volume
        emptyDir: {}                 # 使用emptyDir作为日志目录
```

资源需求和 QoS 保证

设置为garunteed级别。

CPU

按需设定，这里设定为4个CPU

内存

根据制作Dockerfile时启动JAVA进程的堆内存相应设置，设置为堆内存大小（1G）的1.5倍。

```
          resources:
            limits:
              memory: 1.5Gi
              cpu: 4
            requests:
              memory: 1.5Gi
              cpu: 4
```

亲和性设置

设置成副本之间不可以位于统一个节点上。

```
    spec:
      affinity:
        podAntiAffinity:  # 指定是nodeAntiAffinity
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
                - key: app
                  operator: In
                  values:
                    - cardbill
            topologyKey: kubernetes.io/hostname  # 在节点上寻找特定label的pod
```

探活与优雅启动

我的理解，探活就是优雅启动。

> 没有时间找如何使用tini的方式了。

SpringBoot 判断是否是 kubernetes 环境的逻辑很简单，检查是否有 `*_SERVICE_HOST` 和 `*_SERVICE_PORT` 这两个环境变量（需要spring boot 2.3 支持）。

在 spring boot 2.3 中引入了容器探针，也就是增加了 `/actuator/health/liveness` 和 `/actuator/health/readiness` 这两个健康检查路径，对于部署在 k8s 中的应用，spring-boot-actuator 将通过这两个路径自动进行健康检查。如果应用运行在 k8s 环境，这些健康检查自动启动，可以配置 `management.endpoint.health.probes.enabled=true` 在任何环境中启用他们。所以需要在application.properties中添加如下配置：

```
management.endpoint.health.probes.enabled=true
```

- Spring Boot 启动过程中的K8S探针如下

Startup phase	LivenessState	ReadinessState	HTTP server	Notes
Starting	`BROKEN`	`REFUSING_TRAFFIC`	Not started	Kubernetes checks the "liveness" Probe and restarts the application if it takes too long.
Started	`CORRECT`	`REFUSING_TRAFFIC`	Refuses requests	The application context is refreshed. The application performs startup tasks and does not receive traffic yet.
Ready	`CORRECT`	`ACCEPTING_TRAFFIC`	Accepts requests	Startup tasks are finished. The application is receiving traffic.

- Spring Boot 关闭过程中的K8S探针如下：

Shutdown phase	Liveness State	Readiness State	HTTP server	Notes
Running	`CORRECT`	`ACCEPTING_TRAFFIC`	Accepts requests	Shutdown has been requested.
Graceful shutdown	`CORRECT`	`REFUSING_TRAFFIC`	New requests are rejected	If enabled, graceful shutdown processes in-flight requests.
Shutdown complete	N/A	N/A	Server is shut down	The application context is closed and the application is shut down.

```
          livenessProbe:
            httpGet:
              path: /actuator/health/liveness
              port: 9090
            initialDelaySeconds: 10
            failureThreshold: 10
            timeoutSeconds: 10
            periodSeconds: 5
          readinessProbe:
            httpGet:
              path: /actuator/health/readiness
              port: 9090
            initialDelaySeconds: 10
            timeoutSeconds: 10
            periodSeconds: 5
```

验证如下。在工程中增加如下依赖：

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

然后使用docker验证（注意下面特别设置的两个环境变量）：

```
sudo docker run -p 9090:9090 --name cardbill -e
KUBERNETES_SERVICE_HOST=10.96.0.2 \
        -e KUBERNETES_SERVICE_PORT=6443  \
        -v '/home/xiaokai/envs/cloud native/job/module8/config':/app/config \
        -d cardbill:1.0
```

校验两个探针端点：

```
$ curl http://127.0.0.1:9090//actuator/health/liveness
{"status":"UP"}

$ curl http://127.0.0.1:9090//actuator/health/readiness
{"status":"UP"}
```

优雅终止

> 来源于“SpringBoot 2.3 新特性之优雅停机，这波操作太秀了！”；

在最新的 spring boot 2.3 版本，内置此功能，不需要再自行扩展容器线程池来处理，目前 spring boot 嵌入式支持的 web 服务器（Jetty、Reactor Netty、Tomcat 和 Undertow）以及反应式和基于 Servlet 的 web 应用程序都支持优雅停机功能。

当使用server.shutdown=graceful启用时，在 web 容器关闭时，web 服务器将不再接收新请求，并将等待活动请求完成的缓冲期。设置如下：

```
#优雅关机
server.shutdown=graceful
#缓冲器最大等待时间
spring.lifecycle=timeout-per-shutdown-phase: 30s
```

优雅终止涉及到两个类：

- org.srpingframework.boot.web.server.Shutdown;
- org.srpingframework.boot.web.server.WebServer;

验证一下。可以看到日志输出了`o.s.b.w.e.tomcat.GracefulShutdown        : Graceful shutdown complete`。

```
$ sudo docker exec -it 130b6d9a611e bash
[sudo] xiaokai 的密码：
root@130b6d9a611e:/app# ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  7 03:45 ?        00:00:43 java -server -Xmx1G -Xms1G -
XX:+UseG1GC -XX:MaxGCPauseMillis=20 -Djava.security.egd=file:/dev/./uran
root         164       0  0 03:54 pts/1    00:00:00 bash
root         171     164  0 03:55 pts/1    00:00:00 ps -ef
root@130b6d9a611e:/app# kill -2 1
# 容器直接退出
# 在本地查看日志
$ sudo docker  logs -f -t 130b6d9a611e
....
2021-12-02T03:46:36.987738859Z 2021-12-02 03:46:36.987  INFO 1 --- [nio-9090-exec-2] o.s.web.servlet.DispatcherServlet        : Completed initialization in 11 ms
2021-12-02T03:55:33.854312612Z 2021-12-02 03:55:33.853  INFO 1 --- [.ShutdownThread] com.hazelcast.instance.impl.Node         : [127.0.0.1]:5801 [external_support] [4.1.1] Running shutdown hook... Current state: ACTIVE
2021-12-02T03:55:33.859609110Z 2021-12-02 03:55:33.859  INFO 1 --- [extShutdownHook] o.s.b.w.e.tomcat.GracefulShutdown        : Commencing graceful shutdown. Waiting for active requests to complete
2021-12-02T03:55:33.871174381Z 2021-12-02 03:55:33.870  INFO 1 --- [tomcat-shutdown] o.s.b.w.e.tomcat.GracefulShutdown        : Graceful shutdown complete
2021-12-02T03:55:33.921879537Z 2021-12-02 03:55:33.921  INFO 1 --- [extShutdownHook] o.s.s.concurrent.ThreadPoolTaskExecutor  : Shutting down ExecutorService 'applicationTaskExecutor'
```

但这个方案最大的问题是它支持的是SIGINT而不是SIGTERM信号！看来还需要转化一下信号！还需要进一步研究。

最终的Deployment文件

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cardbill-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: cardbill
  strategy:
    rollingUpdate:
      maxSurge: 25%  # 发布新版本的时候，先用新版本启动maxSurge比例的POD
      maxUnavailable: 25%  # 如果不Ready的POD达到了maxUnavailable，发布升级就停止直到有人介入
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: cardbill
    spec:
      affinity:
        podAntiAffinity:  # 指定是nodeAntiAffinity
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: app
                operator: In
                values:
                  - cardbill
```

```yaml
            topologyKey: kubernetes.io/hostname  # 在节点上寻找特定label的pod
      containers:
        - name: cardbill
          image: dxktt/cardbill
          imagePullPolicy: IfNotPresent
          volumeMounts:
          - mountPath: /app/config   # spring boot应用优先读取jar包所在目录的config子
目录下的配置
            name: config-volume
            readOnly: true
          - mountPath: /app/logs     # 日志目录
            name: log-volume
          livenessProbe:
            httpGet:
              path: /actuator/health/liveness
              port: 9090
            initialDelaySeconds: 30
            failureThreshold: 2
            timeoutSeconds: 10
            periodSeconds: 5
          readinessProbe:
            httpGet:
              path: /actuator/health/readiness
              port: 9090
            initialDelaySeconds: 30
            failureThreshold: 2
            timeoutSeconds: 10
            periodSeconds: 5
          env:
            - name: cardBillBondPrivatekey
              valueFrom:
                secretKeyRef:
                  name: card-secret
                  key: cardBillBondPrivatekey
            - name: cardBillBondPublickey
              valueFrom:
                secretKeyRef:
                  name: card-secret
                  key: cardBillBondPublickey
            - name: cardBillBondAesSaltKey
              valueFrom:
                secretKeyRef:
                  name: card-secret
                  key: cardBillBondAesSaltKey
            - name: cardBillBondAeskey
              valueFrom:
                secretKeyRef:
                  name: card-secret
                  key: cardBillBondAeskey
            - name: cardBillBondVectorKey
              valueFrom:
                secretKeyRef:
                  name: card-secret
                  key: cardBillBondVectorKey
          resources:
            limits:
              memory: 1.5Gi
              cpu: 4
            requests:
              memory: 1.5Gi
              cpu: 4
      volumes:
      - name: config-volume
        configMap:
          name: card-config
      - name: log-volume
        emptyDir: {}                # 使用emptyDir作为日志目录
      restartPolicy: Always
```

运行

先以一个副本数量运行，可以按到运行成功

```
$ kubectl create -f card-secret.yaml
$ kubectl create configmap card-config --from-file=application.properties
$ kubectl create -f card-deploy.yaml
$ k get deploy
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
cardbill-deployment   1/1     1            1           71s
```

增加一个副本数

```
$ k scale deploy cardbill-deployment --replicas=2
deployment.apps/cardbill-deployment scaled

$ k get deploy
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
cardbill-deployment   1/2     2            1           7m25s

$ k get po
NAME                                   READY   STATUS    RESTARTS   AGE
cardbill-deployment-6df664fccb-whmw8   0/1     Pending   0          55s
cardbill-deployment-6df664fccb-x8sml   1/1     Running   0          8m14s
```

可以看到第二个POD为pending状态，因为POD有反亲和性的设置，一个节点只能运行一个POD

```
$ k describe po cardbill-deployment-6df664fccb-whmw8
...

Events:
  Type     Reason            Age   From               Message
  ----     ------            ----  ----               -------
  Warning  FailedScheduling  113s  default-scheduler  0/1 nodes are available: 1
node(s) didn't match pod affinity/anti-affinity, 1 node(s) didn't match pod
anti-affinity rules.
  Warning  FailedScheduling  113s  default-scheduler  0/1 nodes are available: 1
node(s) didn't match pod affinity/anti-affinity, 1 node(s) didn't match pod
anti-affinity rules.
...
```

作业二

> 除了将 httpServer 应用优雅的运行在 Kubernetes 之上，我们还应该考虑如何将服务发布给对内和对外的调用方。
> 来尝试用 Service, Ingress 将你的服务发布给集群外部的调用方吧
> 在第一部分的基础上提供更加完备的部署 spec，包括（不限于）
>
> - Service
> - Ingress
>
> 可以考虑的细节
>
> - 如何确保整个应用的高可用
> - 如何通过证书保证 httpServer 的通讯安全

Service

创建clusterIP类型的service

```
apiVersion: v1
kind: Service
metadata:
  name: card-bill
spec:
  type: ClusterIP
  ports:
    - port: 80
      protocol: TCP
      targetPort: 9090
  selector:
    app: cardbill

```

Ingress

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: gateway
  annotations:
    kubernetes.io/ingress.class: "nginx"  #当有多个ingress Controller时指定交给NGINX
的ingress Controller处理
spec:
  tls:
    - hosts:
        - example.com
      secretName: example-tls  #配置TLS服务是，KEY和CERT去这个secret对象里寻找
  rules:   # 规则
    - host: example.com  #规则生效的域名
      http:
        paths:
          - path: "/cardBill"
            pathType: Prefix   # 最终的规则为访问“example.com/cardBill/**”时转发到名
为nginx的服务
            backend:
              service:
                name: card-bill
```

```yaml
                  port:
                    number: 80
```

secret example-tls如下：

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: example-tls
data:
  tls.crt:
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURMVENDQWhXZ0F3SUJBZ0lVVnVFWW11VHdxYXVF
QjlxaGNSQUNRZDEyS0FrdORRWUpLb1pJaHZjTkFRRUwKQlFBd0pqRVRNQkVHQTFVRUF3d0tZMjVqWVcx
d0xtTnZiVEVQTUEwR0ExVUVDZ3dHWTI1allXMXdNQjRYRFRJeApNVEV3TWpFeE1URXdNVm9YRFRJeU1U
RXdNakV4TVRFd01Wb3dKakVUTUJFR0ExVUVBd3dLWTI1allXMXdMbU52CmJURVBNQTBHQTFVRUNnd0dZ
MjVqWVcxd01JSUJJakFOQmdrcWhraUc5dzBCQVFFRkFBT0NBUThBTUlJQkNnS0MKQVFFQTVJUlY1TGJI
T0tEVjE1NEZBUk55cE1IUzhKaUIrSXdBT3ZNQzR0R0xKdUg4SndsV3JnaVR4dW9JQjJDegpwVUp2RjZZ
ck9PVGlsYTdkc1NMNTVLMWZuY0wxZlR6VVIrSk9ydkVPbndORjJRSEZKSU1vMzBqNmZsNmVWSFdqClhk
M2tNMDc0ZnNKYkc0ZVBCR2YzYmNVamNoWjlPZ09JQXpCdXpSbDhzSlRSZnJuRklOeUxhUjZZOUtRRGNY
dzAKSWU1VkIyM2Q2ZGRJcDJvMzhtS2FQR1RQU0hhZDVUSWMvaXdscmlzaGxNZ2VBcEZLN0YyL0ZURDlN
cytzWE94Mwp6Z2FSN2tUR0xvQzc3N1pQNTkvMVplUmxWSEora1djY2dESHpsQXVTZy9JemQ5NTZJUFhZ
TTNkTWZ2alNpZXZnCmJsQkJmNnkvcFdiVi9TVFdsdE4wSk9iMFZRSURBUUFCbzFNd1VUQWRCZ05WSFE0
RUZnUVV2RStkaDlJUXVZVWIKdkxYbXlpMTQwUGZtcC9zd0h3WURWUjBqQkJnd0ZvQVV2RStkaDlJUXVZ
VWJ2TFhteWkxNDBQZm1wL3N3RHdZRApWUjBUQVFIL0JBVXdBd0VCL3pBTkJna3Foa2lHOXcwQkFRc0ZB
QU9DQVFFQWNvYzVVaHRlNVNkWWtXaXArbnAwCnp2RnQ2YytENUN4U3Q5aXNYT3QrY3MxQkw5elRKRkNW
dFl2TlIrUXhLWTdnaEh0NkphUjByRFpPa3VCY1pXbkwKTjVqOVRWWng4OG1YbEJMMlFZT21MWjNUcUlV
TnlJMjAxUGxIZmZMb1BxaHZTeGZ4bGg4VnA4NFNaV1lucU5KMQo0cHF4WDFiYk8vZnM3QVNEaVRpZEow
MFM4UjJsZTc0SEU5L0dBbnpuMTEvWTE0L3J1NjRuR2QxdVJQb2JTQUlCCmI1UTRjYUtuOWxBMmFhMXVu
ZmZsZmFQb292eFN1ZHI1dEYzeeUdrTERSaUNNdURidjNsNXBkbmdpWHdBOGZmblgKTVB1aFB0UEtXLzdt
QzZFUUF4Y1BWa2R6UjRKbDQzM2lYYnJ0bXRONW5tRXJ3NEF6dUY4ZGcxWUN4ZU04MzltcQpmUT09Ci0t
LS0tRU5EIENFUlRJRklDQVRFLS0tLS0K
  tls.key:
LS0tLS1CRUdJTiBQUklWQVRFIEtFWS0tLS0tCk1JSUV2UUlCQURBTkJna3Foa2lHOXcwQkFRRUZBQVND
Qktjd2dnU2pBZ0VBQW9JQkFRRGtoRlhrdHNjNG9OWFgKbmdVQkUzS2t3ZEx3bUlINGpBQTY4d0xpMFlz
bTRmd25DVmF1Q0pQRzZnZ0hZTE9sUW04WHBczQ1T0tWcnQyeApJdm5rclYrZHd2VjlQTlJINGs2dThR
NmZBMFhaQWNVa2d5amZTUHArWHA1VWRhTmQzZVF6VHZoK3dsc2JoNDhFClovZHR4U055Rm4wNkE0Z0RN
RzdOR1h5d2xORit1Y1VpM0l0cEhwajBwQU54ZkRRaDdsVUhiZDNwMTBpbmFqZnkKWXBvOFpNOUlkcDNs
TWh6K0xDV3VLeUdveUI0Q2tVcnNYYjhWTVAweXo2eGM3SGZPQnBIdVJNWXVnTHZ2dGsvbgozL1ZsNUdW
VWNuNlJaeHlBTWZPVUM1S0Q4ak4zM25vZzlkZ3pkMHgrK05LSjYrQnVVRUYvckwrbFpOWDlKTmFXCjAz
UWs1dlJWQWdNQkFBRUNnZ0VBVUd5R3NrVGxubmQwYXpdzlhejdoeUt6ODNzcEd1bGkxMXhaWXF4dXRk
T0kKNDlBMGtuRndXT3hhd1FYMms2M3EzVDdkTFZ2WXB1ZHhISHQ3eVZCL08wMjNDa21UU0cxTVZlTit5
dFhqQ2puRwpRVkJyM1JHWkgwcDduS3YrUC9YczcyWFdyUDRJQkl0VCtUQUI4NzhTOTM4VXVZc3Vuamk0
ZGpTSEhycHhkSkNRCm1JcG9GNWpCUzg4WjdpNWxQTHVXUTVYK2pmcGVqaU56dDlqcWJaL1BzRFcrZUNR
eUNkRlVBRFpwNGZrZTdWS1oKc2I0U3JlUlA4bmh5ejZxT29BcVkvOHVpcUpRbVVVEWldyUzRM
LzZlbFFGa2cyK1lhekhsV0xuTW5mbQpSaENlU3EyOGdkUlR2eHpCUS9Ta0ZBcFZVd0lzdmpGcFFXTZz
RnRMUVFLQmdRRDNFSHhvK2NYQjAwV1gzTW1ZClRQWUNEeEMxNlhTemlqbkdXTUZmdURnN3BkejJ0c1h1
MEJDRlFLN1AwV0cxY1lJMTJDbXM3V0lIZi9tRWJORDgKK3Y4czllZkhFamZGNklHREFUMGxEMlhGS1lp
ZC9MRDdFQmFKejBRcnhvaVBTS0ZqMlBOeHVLRkhCM3hmQkZ1cQpKV2pSUHRxaDEwOVF3Qzl0bGRMNXFl
Tk1VUUtCZ1FFc3lCOXJnQVQveW9PalovRlFWYXBvbjE1aS9reWZZdGpPCm5qcVhxaStQRXIyS05zdUU5
cXpUb2YyZldrSVVwWk5DemFOYlNhTjBCUHpKNnB2MW1IZTlIVmIyaUxkZUw2ekseUZKcUhGVkVhcGNz
OHo5dFhhVldJVGlrYW1MZ0l6ZWVPUnJDUVRXRVzenZjQjFHNmhYMFRXY0OvMHl5NkIyYgpJL2c3R1Mw
YXhRS0JnQnU3SWZ1MmJWZ0FHc09jNkpQTHI2RXpoN0NqYjVIblpleTVjWnNJMW1iN2l2MjJMaWxMCitV
NjduK043b1BmNHNhRFJqbnYyZVJaV2F1OU9OM2J0eFU2S2OyVmd5aG5RcXhqRlB0TzJFcm82bXpjQWNl
ejgKTlV0cWxFQkZuSFpZdEtTOGFUYk9mbXUyajJNcTNnOE8rK0RncVRHWk5USGZJSE93YVduVWc4ckJB
b0dBVU9FaQo2bU4yVWJGcE9iM2RqZVZWS2h1VjhCYVJNYmhmK21QTis5UmtIbWoyV1duUU0p2N2psZjYx
VldOTlRBVyt3WWpnCmplUWZjeGZwQ1VlY01rMzhTSnJuQTVzN2wyNk5oVTdiNStiNXNUNy9tSmtXUjFN
L2tLWVVaUVQ1OVRuU1c3ZUIKem16YjA2RkF5MkR1ZnpTaWZ5cVpVclU2QzdxQnNtYWMrZ0xsaDBrQ2dZ
RUFpdTYwR1d5Mnl4Tm45aXNjalZYbwpDUFNSbmtFM3MzM05YcnJwLzArbVZZMdVVrUnRFcjBqWi9CTFl2
NlRDMVhuaGdXMlZqdXZkWU9NVV0WEM0M2JOClllXYXArUWMyYmZrRTJsWjZhdllNYXpEQ014Tzh5a3Zy
bWgwTWhHbDRkeXQvMWdCZnR1REphRVErQU5HaFEwRGMKbzFlWGlJTkh2aGdhTTN4YkdLREdMWFk9Ci0t
LS0tRU5EIFBSSVZBVEUgS0VZLS0tLS0K
type: kubernetes.io/tls
```

测试

```
$ k create -f card-service.yaml
service/card-bill created

$ k get service
NAME                       TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
      AGE

$ k create -f card-ingress.yaml
ingress.networking.k8s.io/gateway created

$ k get ingress
Warning: extensions/v1beta1 Ingress is deprecated in v1.14+, unavailable in
v1.22+; use networking.k8s.io/v1 Ingress
NAME      CLASS    HOSTS         ADDRESS   PORTS     AGE
gateway   <none>   example.com             80, 443   4s

$ k get service card-bill
NAME        TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
card-bill   ClusterIP   10.104.84.27   <none>        80/TCP    99s

$ curl 10.104.84.27
{"timestamp":"2021-12-02T05:32:06.501+00:00","status":404,"error":"Not
Found","message":"","path":"/"}
```