k8s基础篇-服务发布

Label && Selector

当Kubernetes对系统的任何API对象如Pod和节点进行“分组”时，会对其添加Label（key=value形式的“键-值对”）用以精准地选择对应的API对象。而Selector（标签选择器）则是针对匹配对象的查询方法。注：键-值对就是key-value pair。
例如，常用的标签tier可用于区分容器的属性，如frontend、backend；或者一个release_track用于区分容器的环境，如canary、production等。

如何定义Label

应用案例：

公司与xx银行有一条专属的高速光纤通道，此通道只能与192.168.7.0网段进行通信，因此只能将与xx银行通信的应用部署到192.168.7.0网段所在的节点上，此时可以对节点进行Label（即加标签）：

```
# 给k8s-node02节点打标签
[root@k8s-master01 ~]# kubectl label node k8s-node02 region=subnet7
node/k8s-node02 labeled

# 查找刚刚打标签的节点，通过Selector对其筛选
[root@k8s-master01 ~]# kubectl get no -l region=subnet7
NAME          STATUS   ROLES    AGE   VERSION
k8s-node02    Ready    <none>   44h   v1.20.0

# 最后，在Deployment或其他控制器中指定将Pod部署到该节点 *****
containers:
  ......
dnsPolicy: ClusterFirst
nodeSelector:
  region: subnet7               # 指定刚刚我们打的标签
restartPolicy: Always
......


# 可以用同样的方式对Service进行Label
[root@k8s-master01 ~]# kubectl label svc canary-v1 -n canary-production env=canary
version=v1
service/canary-v1 labeled

# 查看Labels：
[root@k8s-master01 ~]# kubectl get svc -n canary-production --show-labels

# 还可以查看所有Version为v1的svc
```

```
kubectl get svc --all-namespaces -l version=v1

# 其他资源的Label方式相同
```

Selector条件匹配

Selector主要用于资源的匹配，只有符合条件的资源才会被调用或使用，可以使用该方式对集群中的各类资源进行分配。

```
# 假如对Selector进行条件匹配，目前已有的Label如下
[root@k8s-master01 ~]# kubectl get no --show-labels
NAME           STATUS   ROLES    AGE   VERSION   LABELS
k8s-master01   Ready    matser   44h   v1.20.0
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kube
rnetes.io/hostname=k8s-master01,kubernetes.io/os=linux,node-
role.kubernetes.io/matser=,node.kubernetes.io/node=
k8s-master02   Ready    <none>   44h   v1.20.0
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kube
rnetes.io/hostname=k8s-master02,kubernetes.io/os=linux,node.kubernetes.io/node=
k8s-master03   Ready    <none>   44h   v1.20.0
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kube
rnetes.io/hostname=k8s-master03,kubernetes.io/os=linux,node.kubernetes.io/node=
k8s-node01     Ready    <none>   44h   v1.20.0
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kube
rnetes.io/hostname=k8s-node01,kubernetes.io/os=linux,node.kubernetes.io/node=
k8s-node02     Ready    <none>   44h   v1.20.0
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kube
rnetes.io/hostname=k8s-
node02,kubernetes.io/os=linux,node.kubernetes.io/node=,region=subnet7

# 选择app为reviews或者productpage的svc
[root@k8s-master01 ~]# kubectl get svc -l  'app in (details, productpage)' --show-
labels
NAME          TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE     LABELS
details       ClusterIP   10.99.9.178      <none>        9080/TCP   45h     app=details
nginx         ClusterIP   10.106.194.137   <none>        80/TCP     2d21h
app=productpage,version=v1
productpage   ClusterIP   10.105.229.52    <none>        9080/TCP   45h
app=productpage,tier=frontend

# 选择app为productpage或reviews但不包括version=v1的svc
[root@k8s-master01 ~]# kubectl get svc -l  version!=v1,'app in (details, productpage)'
--show-labels
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE   LABELS
details       ClusterIP   10.99.9.178     <none>        9080/TCP   45h   app=details
productpage   ClusterIP   10.105.229.52   <none>        9080/TCP   45h
app=productpage,tier=frontend
```

```
# 选择labelkey名为app的svc
[root@k8s-master01 ~]# kubectl get svc -l app --show-labels
NAME          TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE     LABELS
details       ClusterIP   10.99.9.178      <none>        9080/TCP   45h     app=details
nginx         ClusterIP   10.106.194.137   <none>        80/TCP     2d21h
app=productpage,version=v1
productpage   ClusterIP   10.105.229.52    <none>        9080/TCP   45h
app=productpage,tier=frontend
ratings       ClusterIP   10.96.104.95     <none>        9080/TCP   45h


[root@k8s-master01 ~]# kubectl get po -A --show-labels
NAMESPACE              NAME                                       READY   STATUS
RESTARTS   AGE   LABELS
default                busybox                                    1/1     Running   1
         14h   <none>
default                nginx-66bbc9fdc5-8sbxs                     1/1     Running   1
         14h   app=nginx,pod-template-hash=66bbc9fdc5
kube-system            calico-kube-controllers-5f6d4b864b-kxtmq   1/1     Running   1
         14h   k8s-app=calico-kube-controllers,pod-template-hash=5f6d4b864b
kube-system            calico-node-6fqbv                          1/1     Running   1
         14h   controller-revision-hash=5fd5cdd8c4,k8s-app=calico-node,pod-template-
generation=1
kube-system            calico-node-7x2j4                          1/1     Running   1
         14h   controller-revision-hash=5fd5cdd8c4,k8s-app=calico-node,pod-template-
generation=1
kube-system            calico-node-8p269                          1/1     Running   1
         14h   controller-revision-hash=5fd5cdd8c4,k8s-app=calico-node,pod-template-
generation=1
kube-system            calico-node-kxp25                          1/1     Running   1
         14h   controller-revision-hash=5fd5cdd8c4,k8s-app=calico-node,pod-template-
generation=1
kube-system            calico-node-xlvd9                          1/1     Running   1
         14h   controller-revision-hash=5fd5cdd8c4,k8s-app=calico-node,pod-template-
generation=1
kube-system            coredns-867d46bfc6-5nzq7                   1/1     Running   1
         14h   k8s-app=kube-dns,pod-template-hash=867d46bfc6
kube-system            metrics-server-595f65d8d5-7qhdk            1/1     Running   1
         14h   k8s-app=metrics-server,pod-template-hash=595f65d8d5
kubernetes-dashboard   dashboard-metrics-scraper-7645f69d8c-h9wqd   1/1     Running   1
         14h   k8s-app=dashboard-metrics-scraper,pod-template-hash=7645f69d8c
kubernetes-dashboard   kubernetes-dashboard-78cb679857-6q686       1/1     Running   1
         14h   k8s-app=kubernetes-dashboard,pod-template-hash=78cb679857

[root@k8s-master01 ~]# kubectl get po -A -l 'k8s-app in(metrics-server, kubernetes-
dashboard)'
NAMESPACE              NAME                                    READY   STATUS
RESTARTS   AGE
```

```
kube-system            metrics-server-595f65d8d5-7qhdk          1/1     Running   1
     14h
kubernetes-dashboard   kubernetes-dashboard-78cb679857-6q686    1/1     Running   1
     14h
```

管理标签（Label）

在实际使用中，Label的更改是经常发生的事情，可以使用overwrite参数修改标签。

```
# 修改标签，比如将version=v1改为version=v2
[root@k8s-master01 ~]# kubectl get svc -n canary-production --show-labels
NAME        TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE   LABELS
canary-v1   ClusterIP   10.110.253.62   <none>        8080/TCP   26h
env=canary,version=v1
[root@k8s-master01 canary]# kubectl label svc canary-v1 -n canary-production version=v2
--overwrite
service/canary-v1 labeled
[root@k8s-master01 canary]# kubectl get svc -n canary-production --show-labels
NAME        TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE   LABELS
canary-v1   ClusterIP   10.110.253.62   <none>        8080/TCP   26h
env=canary,version=v2

# 删除标签
[root@k8s-master01 ~]# kubectl label svc canary-v1 -n canary-production version-
service/canary-v1 labeled
[root@k8s-master01 canary]# kubectl get svc -n canary-production --show-labels
NAME        TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE   LABELS
canary-v1   ClusterIP   10.110.253.62   <none>        8080/TCP   26h   env=canary
```

Service

什么是Service

Service可以简单的理解为逻辑上的一组Pod。一种可以访问Pod的策略，而且其他Pod可以通过这个Service访问到这个Service代理的Pod。相对于Pod而言，它会有一个固定的名称，一旦创建就固定不变。

可以简单的理解成访问一个或者一组Pod的时候，先访问service再去访的IP的,service的名称的固定的，不管你怎么重启Pod，Pod的IP怎么改变，都不影响用户的使用

创建一个简单的Service

```
[root@k8s-master01 ~]# cat nginx-svc.yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    app: nginx-svc    #service 标签名字
  name: nginx-svc   #service 固定名字
spec:
  ports:
  - name: http    # Service端口的名称
    port: 80      # Service自己的端口, servicea --> serviceb http://serviceb,
http://serviceb:8080
    protocol: TCP   # UDP TCP SCTP default: TCP
    targetPort: 80  # 后端应用的端口
  - name: https
    port: 443
    protocol: TCP
    targetPort: 443
  selector:
    app: nginx
  sessionAffinity: None
  type: ClusterIP

# 创建一个service
[root@k8s-master01 ~]# kubectl create -f nginx-svc.yaml

[root@k8s-master01 ~]# kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP          19h
nginx-svc    ClusterIP   10.102.13.187   <none>        80/TCP,443/TCP   2m50s
```

使用Service代理k8s外部应用

使用场景：

希望在生产环境中使用某个固定的名称而非IP地址进行访问外部的中间件服务
希望Service指向另一个Namespace中或其他集群中的服务
某个项目正在迁移至k8s集群，但是一部分服务仍然在集群外部，此时可以使用service代理至k8s集群外部的服务

```
# 创建一个类型为external的service（svc），这个svc不会自动创建一个ep
[root@k8s-master01 ~]# vim nginx-svc-external.yaml
apiVersion: v1
kind: Service
metadata:
```

```
  labels:
    app: nginx-svc-external
  name: nginx-svc-external
spec:
  ports:
  - name: http # Service端口的名称
    port: 80 # Service自己的端口, servicea --> serviceb http://serviceb,
http://serviceb:8080
    protocol: TCP # UDP TCP SCTP default: TCP
    targetPort: 80 # 后端应用的端口
  sessionAffinity: None
  type: ClusterIP

# create svc
[root@k8s-master01 ~]# kubectl create -f nginx-svc-external.yaml
service/nginx-svc-external created

# 查看svc
[root@k8s-master01 ~]# kubectl get svc
NAME                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
kubernetes           ClusterIP   10.96.0.1       <none>        443/TCP          6d4h
nginx-svc            ClusterIP   10.96.141.65    <none>        80/TCP,443/TCP   4d3h
nginx-svc-external   ClusterIP   10.109.18.238   <none>        80/TCP           16s
```

```
# 手动创建一个ep，跟上面创建的svc关联起来
[root@k8s-master01 ~]# vim nginx-ep-external.yaml
apiVersion: v1
kind: Endpoints
metadata:
  labels:
    app: nginx-svc-external   #名字要跟svc的一致
  name: nginx-svc-external
  namespace: default
subsets:
- addresses:
  - ip: 220.181.38.148 # baidu
  ports:
  - name: http
    port: 80
    protocol: TCP

# create ep
[root@k8s-master01 ~]# kubectl create -f nginx-ep-external.yaml
endpoints/nginx-svc-external created

# 查看ep (EP后面对应的列表就是可用Pod的列表)
[root@k8s-master01 ~]# kubectl get ep
NAME                 ENDPOINTS
      AGE
```

```
kubernetes           192.168.1.100:6443,192.168.1.101:6443,192.168.1.102:6443
     6d4h
nginx-svc            172.161.125.15:443,172.162.195.15:443,172.169.244.194:443 + 9
more...   4d3h
nginx-svc-external   220.181.38.148:80
      35s

# 访问ep
[root@k8s-master01 ~]# curl 220.181.38.148:80 -I
HTTP/1.1 200 OK
Date: Sat, 26 Dec 2020 16:00:57 GMT
Server: Apache
Last-Modified: Tue, 12 Jan 2010 13:48:00 GMT
ETag: "51-47cf7e6ee8400"
Accept-Ranges: bytes
Content-Length: 81
Cache-Control: max-age=86400
Expires: Sun, 27 Dec 2020 16:00:57 GMT
Connection: Keep-Alive
Content-Type: text/html
```

使用Service反代域名

```
cat > nginx-externalName.yaml  << EOF
apiVersion: v1
kind: Service
metadata:
  labels:
    app: nginx-externalname
  name: nginx-externalname
spec:
  type: ExternalName
  externalName: www.baidu.com
EOF

# create svc
[root@k8s-master01 ~]# kubectl create -f nginx-externalName.yaml
service/nginx-externalname created

# 查看svc
[root@k8s-master01 ~]# kubectl get svc
NAME                 TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)
AGE
kubernetes           ClusterIP      10.96.0.1       <none>          443/TCP
6d4h
nginx-externalname   ExternalName   <none>          www.baidu.com   <none>
27s
```

```
nginx-svc             ClusterIP      10.96.141.65    <none>          80/TCP,443/TCP
4d3h
nginx-svc-external    ClusterIP      10.109.18.238   <none>          80/TCP
18m
```

SVC类型

```
ClusterIP：在集群内部使用，也是默认值

ExternalName：通过返回定义的CNAME别名

NodePort：在所有安装了kube-proxy的节点上打开一个端口，此端口可以代理至后端Pod，然后集群外部可以使用节点的IP地址和NodePort的端口号访问到集群Pod的服务。NodePort端口范围默认是30000-32767

LoadBalancer：使用云提供商的负载均衡器公开服务
```

Ingress

什么是Ingress

```
  通俗来讲，ingress和之前提到的Service、Deployment，也是一个k8s的资源类型，ingress用于实现用域名的方式访问k8s内部应用
  管理对集群中的服务(通常是HTTP)的外部访问的API对象。Ingress可以提供负载平衡、SSL终端和基于名称的虚拟主机
```

Ingress安装

官方： https://kubernetes.github.io/ingress-nginx/deploy/

首先安装helm管理工具

https://helm.sh/docs/intro/install/

```
# 1、下载
[root@k8s-master01 ~]# wget https://get.helm.sh/helm-v3.4.2-linux-amd64.tar.gz

# 2、安装
[root@k8s-master01 ~]# tar -zxvf helm-v3.4.2-linux-amd64.tar.gz
[root@k8s-master01 ~]# mv linux-amd64/helm /usr/local/bin/helm
```

使用helm安装ingress

```
# 1、添加ingress的helm仓库
[root@k8s-master01 ~]# helm repo add ingress-nginx
https://kubernetes.github.io/ingress-nginx
"ingress-nginx" has been added to your repositories

# 2、下载ingress的helm包至本地
[root@k8s-master01 ~]# mkdir /helm_images && cd /helm_images
[root@k8s-master01 helm_images]# helm pull ingress-nginx/ingress-nginx

# 3、更改对应的配置
[root@k8s-master01 helm_images]# tar -zxvf ingress-nginx-3.17.0.tgz && cd ingress-nginx

# 4、需要修改的位置
  a)  Controller和admissionWebhook的镜像地址，需要将公网镜像同步至公司内网镜像仓库
  b)  hostNetwork设置为true
  c)  dnsPolicy设置为 ClusterFirstWithHostNet
  d)  NodeSelector添加ingress: "true"部署至指定节点
  e)  类型更改为kind: DaemonSet
  f)  镜像仓库地址需要改2处
  g)  type: ClusterIP

  修改完成后的文件：

controller:
  image:
    repository: registry.cn-beijing.aliyuncs.com/dotbalo/controller   #此处
    tag: "v0.40.2"
    pullPolicy: IfNotPresent
    runAsUser: 101
    allowPrivilegeEscalation: true
  containerPort:
    http: 80
    https: 443
  config: {}
  configAnnotations: {}
  proxySetHeaders: {}
  addHeaders: {}
  dnsConfig: {}
  dnsPolicy: ClusterFirstWithHostNet  #此处
  reportNodeInternalIp: false
  hostNetwork: true   #此处
  hostPort:
    enabled: false
    ports:
      http: 80
      https: 443
  electionID: ingress-controller-leader
```

```
ingressClass: nginx
podLabels: {}
podSecurityContext: {}
sysctls: {}
publishService:
  enabled: true
  pathOverride: ""
scope:
  enabled: false
tcp:
  annotations: {}
udp:
  annotations: {}
maxmindLicenseKey: ""
extraArgs: {}
extraEnvs: []
kind: DaemonSet #此处
annotations: {}
labels: {}
updateStrategy: {}
minReadySeconds: 0
tolerations: []
affinity: {}
topologySpreadConstraints: []
terminationGracePeriodSeconds: 300
nodeSelector:
  kubernetes.io/os: linux
  ingress: "true"     #此处
livenessProbe:
  failureThreshold: 5
  initialDelaySeconds: 10
  periodSeconds: 10
  successThreshold: 1
  timeoutSeconds: 1
  port: 10254
readinessProbe:
  failureThreshold: 3
  initialDelaySeconds: 10
  periodSeconds: 10
  successThreshold: 1
  timeoutSeconds: 1
  port: 10254
healthCheckPath: "/healthz"
podAnnotations: {}
replicaCount: 1
minAvailable: 1
resources:
  requests:
    cpu: 100m
```

```
    memory: 90Mi
autoscaling:
  enabled: false
  minReplicas: 1
  maxReplicas: 11
  targetCPUUtilizationPercentage: 50
  targetMemoryUtilizationPercentage: 50
autoscalingTemplate: []
keda:
  apiVersion: "keda.sh/v1alpha1"
  enabled: false
  minReplicas: 1
  maxReplicas: 11
  pollingInterval: 30
  cooldownPeriod: 300
  restoreToOriginalReplicaCount: false
  triggers: []
  behavior: {}
enableMimalloc: true
customTemplate:
  configMapName: ""
  configMapKey: ""
service:
  enabled: true
  annotations: {}
  labels: {}
  externalIPs: []
  loadBalancerSourceRanges: []
  enableHttp: true
  enableHttps: true
  ports:
    http: 80
    https: 443
  targetPorts:
    http: http
    https: https
  type: ClusterIP   # 此处
  nodePorts:
    http: ""
    https: ""
    tcp: {}
    udp: {}
  internal:
    enabled: false
    annotations: {}
    loadBalancerSourceRanges: []
extraContainers: []
extraVolumeMounts: []
extraVolumes: []
```

```
  extraInitContainers: []
  admissionWebhooks:
    annotations: {}
    enabled: true
    failurePolicy: Fail
    port: 8443
    certificate: "/usr/local/certificates/cert"
    key: "/usr/local/certificates/key"
    namespaceSelector: {}
    objectSelector: {}
    service:
      annotations: {}
      externalIPs: []
      loadBalancerSourceRanges: []
      servicePort: 443
      type: ClusterIP
    patch:
      enabled: true
      image:
        repository: registry.cn-beijing.aliyuncs.com/dotbalo/kube-webhook-certgen #此处
        tag: v1.3.0
        pullPolicy: IfNotPresent
      priorityClassName: ""
      podAnnotations: {}
      nodeSelector: {}
      tolerations: []
      runAsUser: 2000
  metrics:
    port: 10254
    enabled: false
    service:
      annotations: {}
      externalIPs: []
      loadBalancerSourceRanges: []
      servicePort: 9913
      type: ClusterIP
    serviceMonitor:
      enabled: false
      additionalLabels: {}
      namespace: ""
      namespaceSelector: {}
      scrapeInterval: 30s
      targetLabels: []
      metricRelabelings: []
    prometheusRule:
      enabled: false
      additionalLabels: {}
      rules: []
  lifecycle:
```

```
    preStop:
      exec:
        command:
          - /wait-shutdown
  priorityClassName: ""
revisionHistoryLimit: 10
defaultBackend:
  enabled: false
  name: defaultbackend
  image:
    repository: k8s.gcr.io/defaultbackend-amd64
    tag: "1.5"
    pullPolicy: IfNotPresent
    runAsUser: 65534
    runAsNonRoot: true
    readOnlyRootFilesystem: true
    allowPrivilegeEscalation: false
  extraArgs: {}
  serviceAccount:
    create: true
    name:
  extraEnvs: []
  port: 8080
  livenessProbe:
    failureThreshold: 3
    initialDelaySeconds: 30
    periodSeconds: 10
    successThreshold: 1
    timeoutSeconds: 5
  readinessProbe:
    failureThreshold: 6
    initialDelaySeconds: 0
    periodSeconds: 5
    successThreshold: 1
    timeoutSeconds: 5
  tolerations: []
  affinity: {}
  podSecurityContext: {}
  podLabels: {}
  nodeSelector: {}
  podAnnotations: {}
  replicaCount: 1
  minAvailable: 1
  resources: {}
  autoscaling:
    enabled: false
    minReplicas: 1
    maxReplicas: 2
    targetCPUUtilizationPercentage: 50
```

```
    targetMemoryUtilizationPercentage: 50
  service:
    annotations: {}
    externalIPs: []
    loadBalancerSourceRanges: []
    servicePort: 80
    type: ClusterIP
  priorityClassName: ""
rbac:
  create: true
  scope: false
podSecurityPolicy:
  enabled: false
serviceAccount:
  create: true
  name:
imagePullSecrets: []
tcp: {}
udp: {}


# 5、部署ingress，给需要部署ingress的节点上打标签，这样就能指定要部署的节点了
[root@k8s-master01 ~]# kubectl label node k8s-master03 ingress=true
node/k8s-master03 labeled

# 创建一个ns
[root@k8s-master01 ~]# kubectl create ns ingress-nginx
namespace/ingress-nginx created

# 部署ingress
[root@k8s-master01 ingress-nginx]# helm install ingress-nginx -n ingress-nginx .

# 查看刚刚构建的ingress
[root@k8s-master01 ingress-nginx]# kubectl get  pod -n ingress-nginx

# ingress扩容与缩容，只需要给想要扩容的节点加标签就行，缩容就把节点标签去除即可
[root@k8s-master01 ~]# kubectl label node k8s-master02 ingress=true
node/k8s-master02 labeled

# ingress缩容
[root@k8s-master01 ~]# kubectl label node k8s-master03 ingress-
node/k8s-master03 labeled
```

Ingress入门使用

```
# 创建一个ingress
cat > ingress.yaml << EFO
apiVersion: networking.k8s.io/v1beta1 # networking.k8s.io/v1 / extensions/v1beta1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: "nginx"
  name: example
spec:
  rules: # 一个Ingress可以配置多个rules
  - host: foo.bar.com # 域名配置，可以不写，匹配*，  *.bar.com
    http:
      paths: # 相当于nginx的location配合，同一个host可以配置多个path / /abc
      - backend:
          serviceName: nginx-svc
          servicePort: 80
        path: /
EFO

# 创建
[root@k8s-master01 ~]# kubectl create -f ingress.yaml

 # win配置 hosts
 # 流浪器访问：  http://foo2.bar.com
```

```
# 创建一个多域名ingress
cat ingress-mulDomain.yaml
apiVersion: networking.k8s.io/v1beta1 # networking.k8s.io/v1 / extensions/v1beta1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: "nginx"
  name: example
spec:
  rules: # 一个Ingress可以配置多个rules
  - host: foo.bar.com # 域名配置，可以不写，匹配*，  *.bar.com
    http:
      paths: # 相当于nginx的location配合，同一个host可以配置多个path / /abc
      - backend:
          serviceName: nginx-svc
          servicePort: 80
        path: /
  - host: foo2.bar.com # 域名配置，可以不写，匹配*，  *.bar.com
    http:
      paths: # 相当于nginx的location配合，同一个host可以配置多个path / /abc
      - backend:
```

```
          serviceName: nginx-svc-external
          servicePort: 80
        path: /

[root@k8s-master01 ~]# kubectl replace -f ingress-mulDomain.yaml
[root@k8s-master01 ~]# kubectl get ingress

# win配置 hosts
# 流浪器访问： http://foo2.bar.com
```

HPA

什么是HPA

HPA（Horizontal Pod Autoscaler，水平Pod自动伸缩器）可根据观察到的CPU、内存使用率或自定义度量标准来自动扩展或缩容Pod的数量。HPA不适用于无法缩放的对象，比如DaemonSet
HPA控制器会定期调整RC或Deployment的副本数，以使观察到的平均CPU利用率与用户指定的目标相匹配
HPA需要metrics-server（项目地址：https://github.com/kubernetes-incubator/metrics-server）获取度量指标，由于在高可用集群安装中已经安装了metrics-server，所以本节的实践部分无须再次安装

HPA原理

为什么要使用HPA

在生产环境中，总会有一些意想不到的事情发生，比如公司网站流量突然升高，此时之前创建的Pod已不足以撑住所有的访问，而运维人员也不可能24小时守着业务服务，这时就可以通过配置HPA，实现负载过高的情况下自动扩容Pod副本数以分摊高并发的流量，当流量恢复正常后，HPA会自动缩减Pod的数量

HPA中一些细节的处理

噪声处理：
　　通过上面的公式可以发现，Target的数目很大程度上会影响最终的结果，而在Kubernetes中，无论是变更或者升级，都更倾向于使用Recreate而不是Restart的方式进行处理。这就导致了在Deployment的生命周期中，可能会出现某一个时间，Target会由于计算了Starting或者Stopping的的Pod而变得很大。这就会给HPA的计算带来非常大的噪声，在HPA Controller的计算中，如果发现当前的对象存在Starting或者Stopping的Pod会直接跳过当前的计算周期，等待状态都变为Running再进行计算。

冷却周期：
　在弹性伸缩中，冷却周期是不能逃避的一个话题，很多时候我们期望快速弹出与快速回收，而另一方面，我们又不希望集群震荡，所以一个弹性伸缩活动冷却周期的具体数值是多少，一直被开发者所挑战。在HPA中，默认的扩容冷却周期是3分钟，缩容冷却周期是5分钟。

边界值计算：
　我们回到刚才的计算公式，第一次我们算出需要弹出的容器数目是5，此时扩容后整体的负载是42%，但是我们似乎忽略了一个问题，一个全新的Pod启动会不会自己就占用了部分资源？此外，8%的缓冲区是否就能够缓解整体的负载情况，要知道当一次弹性扩容完成后，下一次扩容要最少等待3分钟才可以继续扩容。为了解决这些问题，HPA引入了边界值△，目前在计算边界条件时，会自动加入10%的缓冲，这也是为什么在刚才的例子中最终的计算结果为6的原因

原理

通过集群内的资源监控系统（metrics-server），来获取集群中资源的使用状态。
根据CPU、内存、以及用户自定义的资源指标数据的使用量或连接数为参考依据，来制定一个临界点，一旦超出这个点，HPA就会自动创建出pod副本
HPA通过定期（定期轮询的时间通过–horizontal-pod-autoscaler-sync-period选项来设置，默认的时间为30秒）通过Status.PodSelector来查询pods的状态，获得pod的CPU使用率。然后，通过现有pods的CPU使用率的平均值（计算方式是最近的pod使用量（最近一分钟的平均值，从metrics-serve中获得）

除以设定的每个Pod的CPU使用率限额）跟目标使用率进行比较，并且在扩容时，还要遵循预先设定的副本数限制：MinReplicas <= Replicas <= MaxReplicas。

计算扩容后Pod的个数：sum(最近一分钟内某个Pod的CPU使用率/量的平均值)/CPU使用上限的整数+1

流程
1、创建HPA资源，设定目标CPU使用率限额，以及最大、最小实例数
2、收集一组中（PodSelector）每个Pod最近一分钟内的CPU使用率，并计算平均值
3、读取HPA中设定的CPU使用限额
4、计算：平均值之和/限额，求出目标调整的实例个数
5、目标调整的实例数不能超过1中设定的最大、最小实例数，如果没有超过，则扩容；超过，则扩容至最大的实例个数
6、回到2，不断循环

实现一个Web服务器的自动伸缩特性

使HPA生效前提：

必须定义 requests参数，必须安装metrics-server

```
# 1、运行hpa资源，名称为php-apache，并设置请求CPU的资源为200m并暴露一个80端口
[root@k8s-master01 ~]#  kubectl run php-apache --image=mirrorgooglecontainers/hpa-example --requests=cpu=200m --expose --port=80
service/php-apache created
pod/php-apache created

# 2、当hpa资源的deployment资源对象的CPU使用率达到20%时，就进行扩容，最多可以扩容到5个
[root@k8s-master01 ~]# kubectl autoscale deployment php-apache --cpu-percent=20 --min=1 --max=5

# 3、确定当前的pod正常运行
[root@master ~]#  kubectl get pod | grep php-apa
php-apache-867f97c8cb-9mpd6    1/1     Running    0          44m
```

模拟消耗php-apache的资源，并验证pod是否会自动扩容与缩容

```
#  新开启多个终端（也可使用node节点），对php-apache的pod进行死循环请求，如下（如果你的系统资源比较充足，可以选择开启多个终端，对pod进行死循环请求
while true; do wget -q -O- 10.97.45.108; done

# 然后查看数量
[root@master ~]# kubectl get pod

# 当停止死循环请求后，也并不会立即减少pod数量，会等一段时间后减少pod数量，防止流量再次激增。

# 至此，HPA实现pod副本数量的自动扩容与缩容就实现了。
```