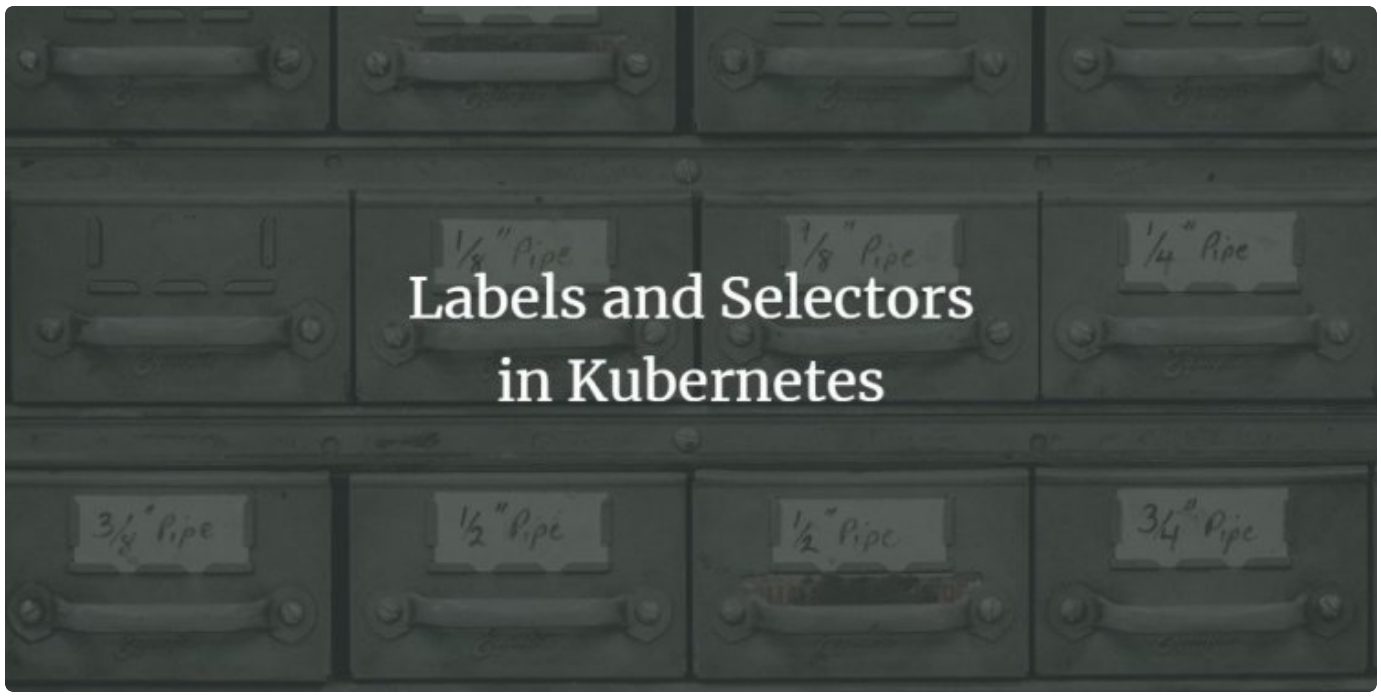
15.什么是标签选择器

> 上一节学习了标签的知识，了解了如何为Pod添加标签、更新标签。这一节我们来学习标签选择器。



如果单看标签的话，似乎作用不太明显，仅仅是附加到资源上的一个标签而已。但是如果将标签与标签选择器配合使用的话，情况就大不一样了。

标签选择器使我们能够从大量的Pod中选择一组具有特定标签的Pod，然后对这些Pod执行统一的操作。

标签选择器其实就是一个选择标准（或选择条件），类似SQL语言中WHERE子句后面的条件，通过判断资源的标签是否满足这个选择标准来过滤资源。

标签选择器是根据如下几个条件来选择资源的：

- 判断资源是否具有包含（或者不包含）某个键（key）的标签
- 判断资源是否具有包含某个键值对的标签
- 判断资源是否具有包含某个键、但是其值与指定的值不相等的标签

1.使用标签选择器(label selector)筛选Pod

当前我们的演示集群中有三个pod：

```
[david@dhr-demo root]$ kubectl get pods --show-labels
NAME             READY   STATUS    RESTARTS   AGE    LABELS
test1            1/1     Running   0          10d    run=test1
test1-manual     1/1     Running   0          3d6h   creation_method=manual
test1-manual-v2  1/1     Running   0          30h    creation_method=manual,env=test
[david@dhr-demo root]$
```

如果我们只想查看标签包含creation_method=manual键值对的Pod，可以执行如下命令：

kubectl get po –l creation_method=manual ––show–labels

```
[david@dhr-demo root]$ kubectl get po -l creation_method=manual --show-labels
NAME             READY   STATUS    RESTARTS   AGE    LABELS
test1-manual     1/1     Running   0          3d6h   creation_method=manual
test1-manual-v2  1/1     Running   0          30h    creation_method=manual,env=test
[david@dhr-demo root]$
```

显示标签包含creattion_method键的Pod：

kubectl get po –l creation_method ––show–labels

```
[david@dhr-demo root]$ kubectl get po -l creation_method --show-labels
NAME             READY   STATUS    RESTARTS   AGE    LABELS
test1-manual     1/1     Running   0          3d8h   creation_method=manual
test1-manual-v2  1/1     Running   0          32h    creation_method=manual,env=test
[david@dhr-demo root]$
```

显示标签不包含creation_method键的Pod：

kubectl get po –l '!creation_method' ––show–labels

```
[david@dhr-demo root]$ kubectl get po -l '!creation_method' --show-labels
NAME    READY   STATUS    RESTARTS   AGE   LABELS
test1   1/1     Running   0          10d   run=test1
```

当然，还有很多其他的标签选择器，比如：

- creation_method!=manual，该选择器只会筛选出标签键为creation_method、值不为manual的所有Pod

  kubectl get po –l creation_method!=abc ––show–labels

  ```
  [david@dhr-demo root]$ kubectl get po -l creation_method!=abc --show-labels
  NAME             READY   STATUS    RESTARTS   AGE    LABELS
  test1            1/1     Running   0          10d    run=test1
  test1-manual     1/1     Running   0          3d8h   creation_method=manual
  test1-manual-v2  1/1     Running   0          32h    creation_method=manual,env=test
  ```

- env in (prod, test)，该选择器只会筛选出标签键为env、值为prod或test的所有Pod

  kubectl get po –l 'env in(prod, test)' ––show–labels

  ```
  [david@dhr-demo root]$ kubectl get po -l 'env in(prod, test)' --show-labels
  NAME             READY   STATUS    RESTARTS   AGE   LABELS
  test1-manual-v2  1/1     Running   0          32h   creation_method=manual,env=test
  [david@dhr-demo root]$
  ```

  需要注意的是使用这种基于集合的标签选择器时，要加上单引号。

- env notin(prod,test)，该选择器会筛选出标签键为env、值不为prod和test的所有Pod

  kubectl get po –l 'env notin(prod,test)' ––show–labels

```
[david@dhr-demo root]$ kubectl get po -l 'env notin(prod,test)' --show-labels
NAME           READY   STATUS    RESTARTS   AGE    LABELS
test1          1/1     Running   0          10d    run=test1
test1-manual   1/1     Running   0          3d9h   creation_method=manual
[david@dhr-demo root]$
```

标签选择器还能包含多个逗号分隔的条件。资源对象必须满足所有这些条件才能被匹配到。

kubectl get po -l creation_method=manual,env=test --show-labels

```
[david@dhr-demo root]$ kubectl get po -l creation_method=manual,env=test --show-labels
NAME             READY   STATUS    RESTARTS   AGE   LABELS
test1-manual-v2  1/1     Running   0          33h   creation_method=manual,env=test
[david@dhr-demo root]$
```

举一个更贴合实际的例子：比如，我们现在有一个集群，里面部署了很多Pod，每个Pod包含一个微服务，如用户服务、订单服务、认证服务等等，其中订单服务都有stable、beta、canary三个版本。如果我们希望只筛选出运行订单服务beta版本的Pod，可以使用app=order,version=beta标签选择器。

2.使用标签选择器约束Pod调度

通常情况下，Kubernetes会将根据资源情况随机地将pod调度到某个工作节点上。但凡事都可能有例外：

比如，我们的Kubernetes集群中有一部分工作节点是使用的传统硬盘技术，另一部分使用的固态硬盘技术，此时可能就需要将某些对硬盘技术有要求的Pod调度的相应的节点上。

再比如，有些Pod需要执行基于GPU的密集型的计算，我们希望将它们只调度到提供了GPU加速功能的工作节点上。

Kubernetes并没有把调度Pod的决定权封闭，它提供标签选择器供我们描述自己的需求，并帮助我们将Pod调度到我们期望的节点。

我们不会去确切地说明希望Pod调度到某个具体的工作节点上，因为我们可能也并不知道集群中有哪些确切的工作节点，另外这也违背Kubernetes对运行在其上的应用隐藏实际基础架构的理念。

但是我们可以描述Pod应该调度到什么样的工作节点上，将这作为对工作节点的要求，并让Kubernetes为我们选择一个满足这些要求的工作节点。

Kubernetes通过使用节点标签和节点标签选择器帮我们实现了这个目标。

为节点指定标签

我们知道，除了Pod，标签还可以作用到其他任何Kubernetes对象上，比如节点（Node）。在向集群中添加节点的时候，一般都会给节点附加一些有用的标签（比如节点提供的硬件类型）以方便Pod的调度。

执行如下命令查看当前环境中的节点：

kubectl get nodes --show-labels

```
[david@dhr-demo root]$ kubectl get nodes --show-labels
NAME       STATUS   ROLES    AGE   VERSION   LABELS
minikube   Ready    master   10d   v1.19.0   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io
netes.io/os=linux,minikube.k8s.io/commit=23aa1eb200a03ae5883dd9d453d4daf3e0f59668,minikube.k8s.io/name=minikube,mini
inikube.k8s.io/version=v1.13.0,node-role.kubernetes.io/master=
[david@dhr-demo root]$
```

可以看到该节点实际上已经包含了很多标签，各标签之间用逗号分隔。

给这个节点添加一个新的标签：

kubectl label node minikube gpu=true

```
[david@dhr-demo ~]$ kubectl get node --show-labels
NAME       STATUS   ROLES    AGE   VERSION   LABELS
minikube   Ready    master   10d   v1.19.0   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,gpu=true,kubernetes.io/arc
ube,kubernetes.io/os=linux,minikube.k8s.io/commit=23aa1eb200a03ae5883dd9d453d4daf3e0f59668,minikube.k8s.io/name=minikube,minikube
08_0700,minikube.k8s.io/version=v1.13.0,node-role.kubernetes.io/master=
[david@dhr-demo ~]$
```

如果要删除这个gpu=true标签，只需要在：key的后面跟上一个减号（-）即可：

kubectl label node minikube gpu-

如果要修改标签，只需要加上--overwrite参数：

kubectl label node minikube gpu=false

当然也可以通过kubectl edit编辑node的配置，修改方式类似vim，保存退出就可以：

kubectl edit node minikube

```
[david@dhr-demo ~]$ kubectl edit node minikube

# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: v1
kind: Node
metadata:
  annotations:
    kubeadm.alpha.kubernetes.io/cri-socket: /var/run/dockershim.sock
    node.alpha.kubernetes.io/ttl: "0"
    volumes.kubernetes.io/controller-managed-attach-detach: "true"
  creationTimestamp: "2020-11-07T08:47:03Z"
  labels:
    beta.kubernetes.io/arch: amd64
    beta.kubernetes.io/os: linux
    gpu: "true"
    kubernetes.io/arch: amd64
    kubernetes.io/hostname: minikube
    kubernetes.io/os: linux
    minikube.k8s.io/commit: 23aa1eb200a03ae5883dd9d453d4daf3e0f59668
    minikube.k8s.io/name: minikube
    minikube.k8s.io/updated_at: 2020_11_07T16_47_08_0700
    minikube.k8s.io/version: v1.13.0
    node-role.kubernetes.io/master: ""
  name: minikube
  resourceVersion: "622993"
  selfLink: /api/v1/nodes/minikube
  uid: ede62fba-33ae-457a-a6ab-475f5f1f20cc
spec: {}
status:
  addresses:
```

使用标签选择器筛选节点：

kubectl get nodes -l gpu=true

```
[david@dhr-demo ~]$ kubectl get nodes -l gpu=true
NAME       STATUS   ROLES    AGE   VERSION
minikube   Ready    master   10d   v1.19.0
```

对于包含很多节点的集群来说，如果想整体显示所有节点的gpu标签值，可以执行：

kubectl get nodes –L gpu

```
[david@dhr-demo ~]$ kubectl get nodes -L gpu
NAME       STATUS   ROLES    AGE   VERSION   GPU
minikube   Ready    master   10d   v1.19.0   true
[david@dhr-demo ~]$
```

将Pod调度到指定的节点

假设现在我们部署一个需要GPU执行工作的Pod。在Pod的YAML文件中添加一个节点标签选择器，这样调度器就只会从那些提供了GPU的节点中选择。

创建一个名为test1–gpu.yaml的文件，内容如下：

vim test1–gpu.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: test1-gpu
spec:
  nodeSelector:
    gpu: "true"
  containers:
  - image: registry.cn-shanghai.aliyuncs.com/david-ns01/test1:1.0
    name: test1
    ports:
    - containerPort: 8080
      protocol: TCP
~
~
```

apiVersion: v1

kind: Pod

metadata:

  name: test1–gpu

spec:

  nodeSelector:

    gpu: "true"

  containers:

  – image: registry.cn–shanghai.aliyuncs.com/david–ns01/test1:1.0

    name: test1

    ports:

    – containerPort: 8080

      protocol: TCP

在创建Pod时，nodeSelector会指示Kubernetes将这个Pod只部署到包含gpu=true标签的节点上。

创建Pod：

kubectl create –f test1–gpu.yaml

```
[david@dhr-demo ~]$ kubectl create -f test1-gpu.yaml
pod/test1-gpu created
[david@dhr-demo ~]$
```

```
[david@dhr-demo ~]$ kubectl get po --show-labels
NAME             READY   STATUS    RESTARTS   AGE     LABELS
test1            1/1     Running   0          10d     run=test1
test1-gpu        1/1     Running   0          40s     <none>
test1-manual     1/1     Running   0          3d11h   creation_method=manual
test1-manual-v2  1/1     Running   0          35h     creation_method=manual,env=test
[david@dhr-demo ~]$
```