**Kubernetes中的标准资源之namespaces**

**目 录**

[第1章 namespaces资源 1](#_Toc185518723)

[1.1 我的kubernetes 1](#_Toc185518724)

[1.2 基于实践理解 1](#_Toc185518725)

[1.3 命名空间作用 3](#_Toc185518726)

[第2章 相关概念的引入 4](#_Toc185518727)

[2.1 资源的API规范 4](#_Toc185518728)

[2.2 资源的apiversion 4](#_Toc185518729)

[2.3 资源的kind 4](#_Toc185518730)

[2.4 资源的metadata之namespace 5](#_Toc185518731)

[2.5 资源的metadata之name 5](#_Toc185518732)

[2.6 资源的metadata之uid 7](#_Toc185518733)

[2.7 资源的metadata之labels 7](#_Toc185518734)

[2.7.1 创建目录 7](#_Toc185518735)

[2.7.2 标签介绍 7](#_Toc185518736)

[2.7.2.1 基本的说明 7](#_Toc185518737)

[2.7.2.2 标签的组成 7](#_Toc185518738)

[2.7.2.3 标签的管理 8](#_Toc185518739)

[2.7.2.4 标签的匹配 8](#_Toc185518740)

[2.7.3 标签实践 11](#_Toc185518741)

[2.7.3.1 创建目录 11](#_Toc185518742)

[2.7.3.2 标签的管理之pods资源对象 11](#_Toc185518743)

[2.7.3.3 标签的管理之nodes资源对象 15](#_Toc185518744)

[2.7.3.4 标签的匹配 17](#_Toc185518745)

[2.8 资源的metadata之annotaions 19](#_Toc185518746)

[2.8.1 创建目录 19](#_Toc185518747)

[2.8.2 注释介绍 19](#_Toc185518748)

[2.8.2.1 基本的介绍 19](#_Toc185518749)

[2.8.2.2 注释的组成 19](#_Toc185518750)

[2.8.2.3 注释的管理 20](#_Toc185518751)

[2.8.3 注释实践 20](#_Toc185518752)

[2.8.3.1 注释的管理之pods资源对象 20](#_Toc185518753)

[2.9 资源的spec和status 23](#_Toc185518754)

# namespaces资源

## 我的kubernetes

我的kubernetes集群是用kubeadm工具部署的，有6个node，6个node上均具备worker node的相关组件（容器运行时、kubelet、kube-proxy）。其中3个是master（有控制平面的相关组件）。

root@master01:~# kubectl get nodes

NAME STATUS ROLES AGE VERSION

master01 Ready control-plane 51d v1.24.3

master02 Ready control-plane 51d v1.24.3

master03 Ready control-plane 51d v1.24.3

node01 Ready <none> 51d v1.24.3

node02 Ready <none> 51d v1.24.3

node03 Ready <none> 51d v1.24.3

## 基于实践理解

namespace资源（简写ns）是kubernetes中的标准资源，且是“非namespace级别”的资源，可用kubectl api-resources命令看到。

root@master01:~# kubectl api-resources | head -1

NAME SHORTNAMES APIVERSION NAMESPACED KIND

root@master01:~# kubectl api-resources | grep namespaces

namespaces ns v1 false Namespace

#

# <== 简写ns、非namespace级别的资源，apiVersion为v1,kind为Namespace

#

使用先编写manifests，再应用manifests以创建ns/lili对象。

* 准备manifests

mkdir $HOME/tools/ns/

cd $HOME/tools/ns/

cat >ns\_lili.yaml<<'EOF'

---

apiVersion: v1

kind: Namespace

metadata:

name: lili

labels:

ns: lili

#spec:

# namespaces资源有spec字段的,但其字段下没有需要额外定义的(ns资源其API规范不复杂)

# 有些字段,其kube-apiserver在"准入控制"会给其进行Mutating(变异)

EOF

* 应用manifests

**## 检查语法**

root@master01:~/tools/ns# kubectl apply -f ns\_lili.yaml --dry-run=client

namespace/lili created (dry run)

**## 应用manifests**

root@master01:~/tools/ns# kubectl apply -f ns\_lili.yaml

namespace/lili created

**## 列出ns/lili对象,并显示labels**

root@master01:~/tools/ns# kubectl get -f ns\_lili.yaml

NAME STATUS AGE

lili Active 33s

root@master01:~/tools/ns# kubectl get ns/lili

NAME STATUS AGE

lili Active 36s

root@master01:~/tools/ns# kubectl get ns/lili --show-labels

NAME STATUS AGE LABELS

lili Active 9m kubernetes.io/metadata.name=lili,ns=lili

使用kubectl工具的create命令也可以创建出ns资源对象。

帮助：kubectl create namespace --help

例如：kubectl create ns lili

说明：无法在创建时指定labels，但可在创建后使用kubectl的label命令来进行添加。

当我们创建一个命名空间（ns资源对象），会自动在里面创建一个sa/default对象和cm/ kube-root-ca.crt对象。

root@master01:~/tools/ns# kubectl -n lili get sa

NAME SECRETS AGE

default 0 19m

# 《== sa资源对象之default

# 《== 此对象就是服务帐户，即服务帐户为default

root@master01:~/tools/ns# kubectl -n lili get cm

NAME DATA AGE

kube-root-ca.crt 1 19m

# 《== cm资源对象之kube-root-ca.crt

# 《== 里面的内容是kubernetes集群的ca证书

一个ns资源对象（命名空间）中存放的是"namespaces级别"的各资源对象，一但删除ns资源对象，那么它里面的相关资源对象也会被删除，所以不要随便删除ns资源对象。那么我在删除ns资源对象前看看里面有哪些资源对象不就行了？是可以的，但你的方法是否能够完全列出某ns资源对象中现有的资源对象呢？

**## kubectl -n <ns资源对象> get all 是无法完全列出来的**

root@master01:~/tools/ns# kubectl -n lili get all

No resources found in lili namespace.

# <== 从前面可知其里面是有sa/default、cm/kube-root-ca.crt对象的

**## 写个”shell脚本“,的格式**

kubectl api-resources --verbs=list --namespaced -o name \

| xargs -n 1 kubectl -n <ns资源对象> get --show-kind --ignore-not-found

**## 确认ns/lili对象中目前有哪些资源对象**

root@master01:~# kubectl api-resources --verbs=list --namespaced -o name | xargs -n 1 kubectl -n lili get --show-kind --ignore-not-found

NAME DATA AGE

configmap/kube-root-ca.crt 1 18h

NAME SECRETS AGE

serviceaccount/default 0 18h

root@master01:~# kubectl api-resources --verbs=list --namespaced -o name | xargs -n 1 kubectl -n lili get --show-kind --ignore-not-found | sed '/^NAME/'d

configmap/kube-root-ca.crt 1 18h

serviceaccount/default 0 18h

前面在创建ns/lili对象时，是先编写manifests，再应用manifests。这里想说的是，一个manifests中可以有多种资源对象（例如a.yaml文件中具备ns/lili资源对象，具备pods/myapp01对象，两者之间用三个减号分隔(---)），但不建议这样做，建议其ns资源对象应该单独一个manifests。如果不这样做，使用kubectl apply -f a.yaml就会出时创建出ns/lili对象，并在ns/lili对象中创建pods/myapp01对象，那如果是kubectl delete -f a.yaml呢，那就会将ns/lili资源对象一同给删除了，而ns/lili对象中可能还会有其它的资源对象，所以这样就存在风险。

## 命名空间作用

用于存放“namespace级别”资源实例化出来的对象，同一命名空间中某资源的对象其name不能相

同，不同命名空间中某资源的对象其name可能相同。

用于逻辑隔离，即并不能完全隔离。比如1：各命名空间中的Pod其实是可以通信的（它们处于同一Pod网络，要想隔离得在网络上想办法）。比如2：svc资源对象会通过标签选择器自身所处命名空间中各Pod上的label。比如3：让不同项目的应用（Pod）放在不同的命名空间中，然后对不同项目的管理员创建相关帐户，让其只能看到其权限允许的命名空间。

其它请参考：<https://kubernetes.io/zh-cn/docs/concepts/overview/working-with-objects/namespaces/>

# 相关概念的引入

## 资源的API规范

kubernetes中大多数资源对象均有apiVersion、kind、metadata、spec、status这四个一级字段，可用kubectl explain <ResourcesName>看到。

kubernetes中不管何种资源，其apiVersion、kind、metadata这三个字段是一定存在的，那么我们在编写某种资源对象的manifests时，我们就得对其进行定义。

kubernetes中大多数资源，均有spec这个一级字段，用于定义资源对象的期望状态，而status字段不需要我们定义，它是资源对象在被创建后的实际状态。

## 资源的apiversion

kubernetes中的各资源的API规范中均有apiVersion这个一级字段（必须得有），可以使用以下方法查看某资源（resource）的apiVersion是什么。

**## 第一种方法：**

root@master01:~# kubectl api-resources | head -2

NAME SHORTNAMES APIVERSION NAMESPACED KIND

bindings v1 true Binding

# 《==可看到bindings这个资源的apiVersion为v1

root@master01:~# kubectl api-resources | grep -w namespaces

namespaces ns v1 false Namespace

# 《==可看到namespaces这个资源的apiVersion为Namespace

**## 第二种方法：以它的为准**

root@master01:~# kubectl explain namespaces | head -4

KIND: Namespace

VERSION: v1

DESCRIPTION:

# 《==可看到pods这个资源的apiVersion为v1

## 资源的kind

kubernetes中的各资源的API中均有kind这个一级字段（必须得有），可以使用以下方法查看某资源（resource）的kind是什么。

**## 第一种方法：**

root@master01:~# kubectl api-resources | head -2

NAME SHORTNAMES APIVERSION NAMESPACED KIND

bindings v1 true Binding

# 《== 可看到bindings这个资源的kind为Binding

root@master01:~# kubectl api-resources | grep -w namespaces

namespaces ns v1 false Namespace

# 《== 可看到namespaces这个资源的kind为Namespace

**## 第二种方法：**

root@master01:~# kubectl explain namespaces | head -4

KIND: Namespace

VERSION: v1

DESCRIPTION:

# 《==可看到namespaces这个资源的kind为Namespace

## 资源的metadata之namespace

kubernetes中的各资源有“namespace级别”"和“非namespace级别”之分，如何确定某资源是否是“namespace级别”的呢？可用kubectl api-resources命令去看。

root@master01:~# kubectl api-resources | head -2

NAME SHORTNAMES APIVERSION NAMESPACED KIND

bindings v1 true Binding

# 《== 可看到bindings这个资源是namespace级别的,因为其NAMESPACE字段的值为true

root@master01:~# kubectl api-resources | grep -w namespaces

namespaces ns v1 false Namespace

# 《== 可看到namespaces这个资源是"非NAMESPACE级别"的。

root@master01:~# kubectl api-resources | grep -w pods

pods po v1 true Pod

# 《== 可看到pods这个资源是namespace级别的。

我们可用kubectl工具列出当前kubernetes中其“namespace级别”、“非namespace级别”的资源分别有哪些。

**## 列出namespace级别的资源**

kubectl api-resources --namespaced=true

**## 列出非namespace级别的资源**

kubectl api-resources --namespaced=false

当某资源是“namespace级别”的，其某资源对象的manifests中metadata.namespace字段是必须的。若某资源是“非namespace级别”的，其资源对象的manifests中metadata.namespace字段不是必须的（指定了也没用）。

## 资源的metadata之name

前面说到了资源有“namespace级别”和“非namespace级别”之分，那么相应的资源对象在相应范围下其name是不能冲突的。

**## 非namespace级别的资源，例如namespaces资源**

namespaes资源对象在整个kubernetes集群中的name不能冲突。ns/a、ns/b

**## namespace级别的资源，例如pods资源**

同一种资源的对象在同一命名空间中，其name不能相同。

同一种资源的对象在不同命名空间中，其name可以相同。

lili名称空间下，pods资源对象的name不能冲突。pods/a、pods/b。

binbin名称空间下，pods资源对象的Name不能冲突。pods/a、pods/b。

不同资源对象的name如何才是有效的呢（除了在所处范围下其name不能冲突，例如name的字符数的范围是多少呢）。kubernetes中各资源对象的name约束有如下四种:

<https://kubernetes.io/zh-cn/docs/concepts/overview/working-with-objects/names/#names>



## 资源的metadata之uid

kubernetes中各资源对象均会有uid的，这个uid不需要我们人为去定义，是由其kubernetes系统在创建各资源对象时为其自动生成的（全局唯一性）。我们可从某资源对象的在线manifests中看到。

**## 查看ns/lili对象的uid**

root@master01:~# kubectl get ns/lili -o=custom-columns=Uid:.metadata.uid

Uid

a11d42cd-57e2-41c4-8b8d-fe66b68a424e

## 资源的metadata之labels

### 创建目录

mkdir -p $HOME/tools/ns/introduce-label/

cd $HOME/tools/ns/introduce-label/

### 标签介绍

#### 基本的说明

<https://kubernetes.io/zh-cn/docs/concepts/overview/working-with-objects/labels/#syntax-and-character-set>

Label（标签）是附加到kubernetes中各资源对象上的键值对（Key=Value），可以有0至多个，例如pods资源其spec.metadata.labels字段就是用来定义pods资源对象的Labels的。

kubernetes中的各资源对象可在被创建前为其附加Labels，当创建后，我们也可以对其进行Labels的在线管理。

kubernetes中有各种资源对象（例如：svc资源对象、pods资源对象）会使用标签进行关联，其引用者（svc资源对象）会使用标签选择器（selector）匹配目标者（pods资源对象）身上的Labels，从而进行关联，以实现具体功能。引用者只需要匹配到目标者身上的某一个标签即算成功，这是标签的特点。引用者若指定多个条件时，这多个条件是and还是or关系呢？这个就得看引用者所用的方法（基于等值匹配、基于集合匹配），不是所有引用者均同时支持所有的方法。

#### 标签的组成

<https://kubernetes.io/zh-cn/docs/concepts/overview/working-with-objects/labels/#syntax-and-character-set>

* 标签的格式

标签是键值对，即Key=Value形式。

* 标签的格式说明

**### Key**

必须且唯一,有前缀和后缀之分,前缀是可选的,若存在前缀,前缀和后缀之间用"/"分隔。

有效前缀：遵循“DNS子域名”

<https://kubernetes.io/zh-cn/docs/concepts/overview/working-with-objects/names/#dns-subdomain-names>

字符数不能超过253个字符。

只能包含字母(小写)、数字，以及破折号 '-' 和点 '.'。

必须以字母、数字开头。

必须以字母、数字结尾。

有效后缀：

字符数必须小于等于63。

只能包含字母(大小写均可)、数字，以及破折号 '-'、下划线'-'、 和点 '.'。

必须以字母、数字开头。

必须以字母、数字结尾。

**## Value**

不是必须的（可以为空）。若不为空，其有效Value如下所示：

其字符数必须小于等于63。

只能包含字母(大小写均可)、数字，以及破折号 '-'、下划线'-'、 和点 '.'。

必须以字母、数字开头。

必须以字母、数字结尾。

#### 标签的管理

* 添加

添加的标签其Key肯定不能与现有标签冲突，不然就是更新标签了。

若采用修改资源对象manifests：指定Key: Value对,再重新应用manifests。

若使用kubectl工具label命令：指定相应资源对象,指定Key=Value对。

* 更新

更新标签是指更新其标签之Key对应的Value，不然就是添加标签了。

若采用修改资源对象manifests：修改其相应标签其Key对应的Value后保存,再应用manifests。

若使用kubectl工具label命令：指定相应资源对象,指定Key=Value对,指定--overwrite=true。

* 删除

若采用修改资源对象manifests：删除其标签的key=Value对后,再重新应用manifests。

若使用kubectl工具label命令：指定相应资源,指定相应标签的Key即可(key-)。

#### 标签的匹配

* 基于等值的匹配，运算符有：等于(=)，不等于(!=)

**## 运算符说明**

等于(=):

例如：key01=value01

表示：存在key01,且key1的值得为value01

不等于(!=):

例如：key02!=value02

表示：不存在key02 或 存在key02,其值不为value02

**## lili名称空间现有资源对象及其拥有的标签**

pods/a对象的标签有：

project=lili

env=prod

tier=frontend

app=a

pods/b对象的标签有：

project=lili

env=prod

tier=frontend

app=b

pods/c对象的标签有：

project=lili

env=prod

tier=backend

app=b

**## 基于等于匹配的等于(=)**

kubectl -n lili get pods -l project=lili,tier=frontend

# <== project=lili和tier=frontend是and关系

# <== 此时匹配到的资源对象为 pods/a、pods/b

kubectl -n lili get pods -l project=lili,tier=backend

# <== project=lili和tier=backend是and关系

# <== 此时匹配到的资源对象为 pods/c

kubectl -n lili get pods -l project=lili

# <== 此时匹配到的资源对象为pods/a、pods/b、pods/c

kubectl -n lili get pods -l tier=frontend

# <== 此时匹配到的资源对象为pods/a、pods/b

kubectl -n lili get pods -l tier=backend

# <== 此时匹配到的资源对象为pods/c

**## 基于等值匹配的不等于(!=)**

kubectl -n lili get pods -l binbin!=haha

# <== 此时匹配到的资源对象为pods/a、pods/b、pods/c

# <== 有binbin这个key,但值不为haha(包括空值)的也会被匹配到。

kubectl -n lili get pods -l tier!=frontend

# <== 此时匹配到的资源对象为pods/c

kubectl -n lili get pods -l tier!=backend

# <== 此时匹配到的资源对象为pods/a、pods/b

kubectl -n lili get pods -l binbin!=,tier!=frontend

# <== binbin!=,tier!=frontend是and关系

# <== 此时匹配到的资源对象为pods/c

* 集合匹配，其运算符有：in、notin、exists

**## 运算符说明**

其value在集合中(in)：

例如：key in (value1,value2)

表示：key存在,且其值必须是集合中的指定的任一value。

其value不在集合中(notin):

例如：key notin (value1,value2)

表示：不存在key 或 key存在,且其值必须是集合中指定的任一value

其key是否存在(exists)

主要判断其key是否存在。

kubectl工具匹配时的写法：key表示key存在,!key表示key不存在。

**## lili名称空间现有资源对象及其拥有的标签**

pods/a对象的标签有：

project=lili

env=prod

tier=frontend

app=a

pods/b对象的标签有：

project=lili

env=prod

tier=frontend

app=b

pods/c对象的标签有：

project=lili

env=prod

tier=backend

app=b

**## 运算符(in)**

kubectl -n lili get pods -l 'tier in (haha,heihei)'

# <== 匹配不到任何的资源对象

kubectl -n lili get pods -l 'tier in (frontend,backend,midd,db)'

# <== 匹配到的资源对象为pods/a、pods/b、pods/c

**## 运算符(notin)**

kubectl -n lili get pods -l 'binbin notin (haha,heihei)'

# <== 匹配到的资源对象为pods/a、pods/b、pods/c

kubectl -n lili get pods -l 'titer notin (midd,db)'

# <== 匹配到的资源对象为pods/a、pods/b、pods/c

**## 运算符(exists)**

kubectl -n lili get pods -l '!binbin'

# <== 匹配到的资源对象为pods/a、pods/b、pods/c

kubectl -n lili get pods -l 'tier'

# <== 匹配到的资源对象为pods/a、pods/b、pods/c

### 标签实践

#### 创建目录

cd $HOME/tools/ns/introduce-label/

mkdir label-manager

mkdir label-match

#### 标签的管理之pods资源对象

这里会为pods/myapp01对象准备多个manifests，各manifests中的对象只有pods/myapp01，各manifests中不同之外就是其Labels，其实只需要写一个manifests，在这一个manifests中进行Labels的更改即可，我这是为了写文档，所以为同一资源对象写了多个manifests。

实践理解的思路为：先逐步应用各manifests并观察。再使用kubectl工具的label为其资源对象进行label的添加、更新、删除操作。

* 编写01.pods\_myapp01.yaml

cd $HOME/tools/ns/introduce-label/label-manager/

cat > 01.pods\_myapp01.yaml <<'EOF'

apiVersion: v1

kind: Pod

metadata:

namespace: lili

name: myapp01

labels:

# 标签的key冲突,以最后一个key的为准

app: "01myapp"

app: "myapp01"

spec:

containers:

- name: myapp01

image: swr.cn-north-1.myhuaweicloud.com/library/nginx:1.16

ports:

- name: http-80

containerPort: 80

EOF

* 编写02.pods\_myapp01-add-label.yaml

cd $HOME/tools/ns/introduce-label/label-manager/

cat > 02.pods\_myapp01-add-label.yaml <<'EOF'

apiVersion: v1

kind: Pod

metadata:

namespace: lili

name: myapp01

labels:

# 标签的key冲突,以最后一个key的为准

app: "01myapp"

app: "myapp01"

tier: "frontend" # <== 这是添加的

project: "lili" # <== 这是添加的

spec:

containers:

- name: myapp01

image: swr.cn-north-1.myhuaweicloud.com/library/nginx:1.16

ports:

- name: http-80

containerPort: 80

EOF

* 编写03.pods\_myapp01-update-label.yaml

cd $HOME/tools/ns/introduce-label/label-manager/

cat > 03.pods\_myapp01-update-label.yaml <<'EOF'

apiVersion: v1

kind: Pod

metadata:

namespace: lili

name: myapp01

labels:

# 标签的key冲突,以最后一个key的为准

app: "01myapp"

app: "myapp01"

tier: "backend" # <== 其value被更新了

project: "lili"

spec:

containers:

- name: myapp01

image: swr.cn-north-1.myhuaweicloud.com/library/nginx:1.16

ports:

- name: http-80

containerPort: 80

EOF

* 编写04.pods\_myapp01-delete-label.yaml

cd $HOME/tools/ns/introduce-label/label-manager/

cat > 04.pods\_myapp01-delete-label.yaml <<'EOF'

apiVersion: v1

kind: Pod

metadata:

namespace: lili

name: myapp01

labels:

# 标签的key冲突,以最后一个key的为准

app: "01myapp"

app: "myapp01"

#tier: "backend" # <== 这是删除的

#project: "lili" # <== 这是删除的

spec:

containers:

- name: myapp01

image: swr.cn-north-1.myhuaweicloud.com/library/nginx:1.16

ports:

- name: http-80

containerPort: 80

EOF

* 逐步应用各manifests，并进行观察

**## 应用01.pods\_myapp01.yaml这个manifests，并观察**

root@master01:~/tools/ns/pod-level/introduce-label/label-manager# kubectl apply -f 01.pods\_myapp01.yaml

pod/myapp01 created

root@master01:~/tools/ns/pod-level/introduce-label/label-manager#

root@master01:~/tools/ns/pod-level/introduce-label/label-manager# kubectl -n lili get pods/myapp01 --show-labels

NAME READY STATUS RESTARTS AGE LABELS

myapp01 1/1 Running 0 7s app=myapp01

**## 应用02.pods\_myapp01-add-label.yaml这个manifests，并观察**

root@master01:~/tools/ns/pod-level/introduce-label/label-manager# kubectl apply -f 02.pods\_myapp01-add-label.yaml

pod/myapp01 configured

root@master01:~/tools/ns/pod-level/introduce-label/label-manager#

root@master01:~/tools/ns/pod-level/introduce-label/label-manager# kubectl -n lili get pods/myapp01 --show-labels

NAME READY STATUS RESTARTS AGE LABELS

myapp01 1/1 Running 0 28s app=myapp01,project=lili,tier=frontend

**## 应用03.pods\_myapp01-update-label.yaml这个manifests，并观察**

root@master01:~/tools/ns/pod-level/introduce-label/label-manager# kubectl apply -f 03.pods\_myapp01-update-label.yaml

pod/myapp01 configured

root@master01:~/tools/ns/pod-level/introduce-label/label-manager#

root@master01:~/tools/ns/pod-level/introduce-label/label-manager# kubectl -n lili get pods/myapp01 --show-labels

NAME READY STATUS RESTARTS AGE LABELS

myapp01 1/1 Running 0 50s app=myapp01,project=lili,tier=backend

**## 应用04.pods\_myapp01-delete-label.yaml这个manifests，并观察**

root@master01:~/tools/ns/pod-level/introduce-label/label-manager# kubectl apply -f 04.pods\_myapp01-delete-label.yaml

pod/myapp01 configured

root@master01:~/tools/ns/pod-level/introduce-label/label-manager#

root@master01:~/tools/ns/pod-level/introduce-label/label-manager# kubectl -n lili get pods/myapp01 --show-labels

NAME READY STATUS RESTARTS AGE LABELS

myapp01 1/1 Running 0 80s app=myapp01

* 使用kubectl工具对pods/myapp01对象进行label的添加、更新、删除

**## 为pods/myapp01对象添加上project=lili,tier=frontend标签**

root@master01:~/tools/ns/pod-level/introduce-label/label-manager# kubectl -n lili label pods/myapp01 project=lili tier=frontend

pod/myapp01 labeled

root@master01:~/tools/ns/pod-level/introduce-label/label-manager#

root@master01:~/tools/ns/pod-level/introduce-label/label-manager# kubectl -n lili get pods/myapp01 --show-labels

NAME READY STATUS RESTARTS AGE LABELS

myapp01 1/1 Running 0 9m39s app=myapp01,project=lili,tier=frontend

**## 为pos/myapp01对象其tier=frontend标签进行更新(肯定是更新其值了,不然就是添加标签了)**

**## 更新成tier=backend**

root@master01:~/tools/ns/pod-level/introduce-label/label-manager# kubectl -n lili label --overwrite=true pods/myapp01 tier=backend

pod/myapp01 labeled

root@master01:~/tools/ns/pod-level/introduce-label/label-manager#

root@master01:~/tools/ns/pod-level/introduce-label/label-manager# kubectl -n lili get pods/myapp01 --show-labels

NAME READY STATUS RESTARTS AGE LABELS

myapp01 1/1 Running 0 12m app=myapp01,project=lili,tier=backend

**## 删除pods/myapp01对象上其project=lili,tier=backend标签**

root@master01:~/tools/ns/pod-level/introduce-label/label-manager# kubectl -n lili label pods/myapp01 project- tier-

pod/myapp01 unlabeled

root@master01:~/tools/ns/pod-level/introduce-label/label-manager#

root@master01:~/tools/ns/pod-level/introduce-label/label-manager# kubectl -n lili get pods/myapp01 --show-labels

NAME READY STATUS RESTARTS AGE LABELS

myapp01 1/1 Running 0 14m app=myapp01

* 清理环境(删除pods/myapp01对象)

kubectl -n lili get pods/myapp01

kubectl -n lili delete pods/myapp01

kubectl -n lili get pods/myapp01

#### 标签的管理之nodes资源对象

我用kubectl工具对node01、node02这两个worker node进行标签的管理。

**## 现在是有标签的(自带的)**

root@master01:~# kubectl get nodes node01 node02 --show-labels

NAME STATUS ROLES AGE VERSION LABELS

node01 Ready <none> 81d v1.24.3 beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostname=node01,kubernetes.io/os=linux

node02 Ready <none> 81d v1.24.3 beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostname=node02,kubernetes.io/os=linux

**## 为其添加上project=wyc,env=prod的标签，并通过标签过滤出相应的nodes**

root@master01:~# kubectl label nodes node01 node02 project=wyc env=prod

node/node01 labeled

node/node02 labeled

root@master01:~#

root@master01:~# kubectl get nodes -l project=wyc,env=prod # 是and的关系

NAME STATUS ROLES AGE VERSION

node01 Ready <none> 81d v1.24.3

node02 Ready <none> 81d v1.24.3

root@master01:~#

root@master01:~# kubectl get nodes -l project=wyc

NAME STATUS ROLES AGE VERSION

node01 Ready <none> 81d v1.24.3

node02 Ready <none> 81d v1.24.3

root@master01:~#

root@master01:~# kubectl get nodes -l env=prod

NAME STATUS ROLES AGE VERSION

node01 Ready <none> 81d v1.24.3

node02 Ready <none> 81d v1.24.3

root@master01:~#

**## 将具备project=wyc标签的各node上的project=wyc标签修改为project=jmsco**

root@master01:~# kubectl label nodes -l project=wyc --overwrite=true project=jmsco

node/node01 labeled

node/node02 labeled

root@master01:~# kubectl get nodes -l project=jmsco,env=prod # 是and的关系

NAME STATUS ROLES AGE VERSION

node01 Ready <none> 81d v1.24.3

node02 Ready <none> 81d v1.24.3

root@master01:~#

root@master01:~# kubectl get nodes -l project=jmsco

NAME STATUS ROLES AGE VERSION

node01 Ready <none> 81d v1.24.3

node02 Ready <none> 81d v1.24.3

root@master01:~#

root@master01:~# kubectl get nodes -l env=prod

NAME STATUS ROLES AGE VERSION

node01 Ready <none> 81d v1.24.3

node02 Ready <none> 81d v1.24.3

**## 将具备project=jmsco,env=prod的node上其project=jmsco,env=prod标签给删除**

root@master01:~# kubectl get nodes -l project=jmsco,env=prod

NAME STATUS ROLES AGE VERSION

node01 Ready <none> 81d v1.24.3

node02 Ready <none> 81d v1.24.3

root@master01:~#

root@master01:~# kubectl label nodes -l project=jmsco,env=prod project- env-

node/node01 unlabeled

node/node02 unlabeled

root@master01:~#

root@master01:~# kubectl get nodes -l project=jmsco,env=prod

No resources found

#### 标签的匹配

这里准备了几个pods资源对象的manifests，你将其应用后，参考27.2.4章节中的示例。

* 准备pods\_myapp01.yaml

cd $HOME/tools/ns/introduce-label/label-match/

cat >pods\_myapp01.yaml<<'EOF'

apiVersion: v1

kind: Pod

metadata:

namespace: lili

name: myapp01

labels:

project: "lili"

env: "prod"

tier: "frontend"

app: "myapp01"

spec:

containers:

- name: myapp01

image: swr.cn-north-1.myhuaweicloud.com/library/nginx:1.16

ports:

- name: http-80

containerPort: 80

EOF

* 准备pods\_myapp02.yaml

cd $HOME/tools/ns/introduce-label/label-match/

cat >pods\_myapp02.yaml<<'EOF'

apiVersion: v1

kind: Pod

metadata:

namespace: lili

name: myapp02

labels:

project: "lili"

env: "prod"

tier: "frontend"

app: "myapp02"

spec:

containers:

- name: myapp02

image: swr.cn-north-1.myhuaweicloud.com/library/nginx:1.16

ports:

- name: http-80

containerPort: 80

EOF

* 准备pods\_myapp03.yaml

cd $HOME/tools/ns/introduce-label/label-match/

cat >pods\_myapp03.yaml<<'EOF'

apiVersion: v1

kind: Pod

metadata:

namespace: lili

name: myapp03

labels:

project: "lili"

env: "prod"

tier: "backend"

app: "myapp03"

spec:

containers:

- name: myapp03

image: swr.cn-north-1.myhuaweicloud.com/library/nginx:1.16

ports:

- name: http-80

containerPort: 80

EOF

## 资源的metadata之annotaions

### 创建目录

mkdir -p $HOME/tools/ns/introduce-annotaions/

cd $HOME/tools/ns/introduce-annotaions/

### 注释介绍

#### 基本的介绍

<https://kubernetes.io/zh-cn/docs/concepts/overview/working-with-objects/annotations/>

annotations（注解）是附加到kubernetes中各资源对象上的键值对（Key=Value），可以有0至多个，例如pods资源其spec.metadata. annotations字段就是用来定义pods资源对象的Labels的。

kubernetes中的各资源对象可在被创建前为其附加annotations，当创建后，我们也可以对其进行annotate的在线管理。

注释的作用是给客户端程序（例如工具和库）能够获取这些元数据信息。

注意：注释没有像标签那样让我们来进行匹配/过滤一说。

#### 注释的组成

和标签label的组成是一样的

* 标签的格式

标签是键值对，即Key=Value形式。

* 标签的格式说明

**### Key**

必须且唯一,有前缀和后缀之分,前缀是可选的,若存在前缀,前缀和后缀之间用"/"分隔。

有效前缀：遵循“DNS子域名”

<https://kubernetes.io/zh-cn/docs/concepts/overview/working-with-objects/names/#dns-subdomain-names>

字符数不能超过253个字符。

只能包含字母(小写)、数字，以及破折号 '-' 和点 '.'。

必须以字母、数字开头。

必须以字母、数字结尾。

有效后缀：

字符数必须小于等于63。

只能包含字母(大小写均可)、数字，以及破折号 '-'、下划线'-'、 和点 '.'。

必须以字母、数字开头。

必须以字母、数字结尾。

**## Value**

不是必须的（可以为空）。若不为空，其有效Value如下所示：

其字符数必须小于等于63。

只能包含字母(大小写均可)、数字，以及破折号 '-'、下划线'-'、 和点 '.'。

必须以字母、数字开头。

必须以字母、数字结尾。

#### 注释的管理

* 添加

添加的标签其Key肯定不能与现有标签冲突，不然就是更新标签了。

若采用修改资源对象manifests：指定Key: Value对,再重新应用manifests。

若使用kubectl工具annotate命令：指定相应资源对象,指定Key=Value对。

* 更新

更新标签是指更新其标签之Key对应的Value，不然就是添加标签了。

若采用修改资源对象manifests：修改其相应标签其Key对应的Value后保存,再应用manifests。

若使用kubectl工具annotate命令：指定相应资源对象,指定Key=Value对,指定--overwrite=true。

* 删除

若采用修改资源对象manifests：删除其标签的key=Value对后,再重新应用manifests。

若使用kubectl工具annotate命令：指定相应资源,指定相应标签的Key即可(key-)。

### 注释实践

#### 注释的管理之pods资源对象

* 准备pods/myapp01对象的manifests

cd $HOME/tools/ns/introduce-annotations/

cat >pods\_myapp01.yaml<<'EOF'

apiVersion: v1

kind: Pod

metadata:

namespace: lili

name: myapp01

labels:

app: "myapp01"

annotations:

# 注释的key冲突,以最后一个为准

author: "chenliang"

author: "chenliang01"

spec:

containers:

- name: myapp01

image: swr.cn-north-1.myhuaweicloud.com/library/nginx:1.16

ports:

- name: http-80

containerPort: 80

EOF

* 应用manifests

**## 应用manifests**

root@master01:~/tools/ns/introduce-annotations# kubectl apply -f pods\_myapp01.yaml

pod/myapp01 created

**## 列出资源对象**

root@master01:~/tools/ns/introduce-annotations# kubectl get -f pods\_myapp01.yaml

NAME READY STATUS RESTARTS AGE

myapp01 1/1 Running 0 22m

**## 显示对象的注释(通过描述信息)**

root@master01:~/tools/ns/introduce-annotations# kubectl describe -f pods\_myapp01.yaml | grep "Annotations:"

Annotations: author: chenliang01

**## 显示注释(通过在线manifests)**

root@master01:~/tools/ns/introduce-annotations# kubectl get -f pods\_myapp01.yaml -o=jsonpath='{.metadata.annotations}' | jq

{

"author": "chenliang01",

"kubectl.kubernetes.io/last-applied-configuration": "{\"apiVersion\":\"v1\",\"kind\":\"Pod\",\"metadata\":{\"annotations\":{\"author\":\"chenliang01\"},\"labels\":{\"app\":\"myapp01\"},\"name\":\"myapp01\",\"namespace\":\"lili\"},\"spec\":{\"containers\":[{\"image\":\"swr.cn-north-1.myhuaweicloud.com/library/nginx:1.16\",\"name\":\"myapp01\",\"ports\":[{\"containerPort\":80,\"name\":\"http-80\"}]}]}}\n"

}

* 注释的添加

root@master01:~/tools/ns/introduce-annotations# kubectl -n lili annotate pods/myapp01 tel=123 tel=321

pod/myapp01 annotated

root@master01:~/tools/ns/introduce-annotations#

root@master01:~/tools/ns/introduce-annotations# kubectl get -f pods\_myapp01.yaml -o=jsonpath='{.metadata.annotations}' | jq

{

"author": "chenliang01",

"kubectl.kubernetes.io/last-applied-configuration": "{\"apiVersion\":\"v1\",\"kind\":\"Pod\",\"metadata\":{\"annotations\":{\"author\":\"chenliang01\"},\"labels\":{\"app\":\"myapp01\"},\"name\":\"myapp01\",\"namespace\":\"lili\"},\"spec\":{\"containers\":[{\"image\":\"swr.cn-north-1.myhuaweicloud.com/library/nginx:1.16\",\"name\":\"myapp01\",\"ports\":[{\"containerPort\":80,\"name\":\"http-80\"}]}]}}\n",

"tel": "321"

}

* 注释的更新

root@master01:~/tools/ns/introduce-annotations# kubectl -n lili annotate pods/myapp01 tel=456 --overwrite

pod/myapp01 annotated

root@master01:~/tools/ns/introduce-annotations#

root@master01:~/tools/ns/introduce-annotations# kubectl get -f pods\_myapp01.yaml -o=jsonpath='{.metadata.annotations}' | jq

{

"author": "chenliang01",

"kubectl.kubernetes.io/last-applied-configuration": "{\"apiVersion\":\"v1\",\"kind\":\"Pod\",\"metadata\":{\"annotations\":{\"author\":\"chenliang01\"},\"labels\":{\"app\":\"myapp01\"},\"name\":\"myapp01\",\"namespace\":\"lili\"},\"spec\":{\"containers\":[{\"image\":\"swr.cn-north-1.myhuaweicloud.com/library/nginx:1.16\",\"name\":\"myapp01\",\"ports\":[{\"containerPort\":80,\"name\":\"http-80\"}]}]}}\n",

"tel": "456"

}

* 注释的删除

root@master01:~/tools/ns/introduce-annotations# kubectl -n lili annotate pods/myapp01 author- tel-

pod/myapp01 annotated

root@master01:~/tools/ns/introduce-annotations#

root@master01:~/tools/ns/introduce-annotations# kubectl get -f pods\_myapp01.yaml -o=jsonpath='{.metadata.annotations}' | jq

{

"kubectl.kubernetes.io/last-applied-configuration": "{\"apiVersion\":\"v1\",\"kind\":\"Pod\",\"metadata\":{\"annotations\":{\"author\":\"chenliang01\"},\"labels\":{\"app\":\"myapp01\"},\"name\":\"myapp01\",\"namespace\":\"lili\"},\"spec\":{\"containers\":[{\"image\":\"swr.cn-north-1.myhuaweicloud.com/library/nginx:1.16\",\"name\":\"myapp01\",\"ports\":[{\"containerPort\":80,\"name\":\"http-80\"}]}]}}\n"

}

* 删除pods/myapp01对象

kubectl get -f pods\_myapp01.yaml

kubectl delete -f pods\_myapp01.yaml

## 资源的spec和status

kubernetes中大多数资源均有spec和status这个两个一级字段，spec此字段用来定义其资源对象的期望状态，status字段是资源对象在被创建后其最终状态的一些信息。