

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

Kubernetes Namespace 和 Label 资源应用

前言：
课程名称：Kubernetes Namespace 与 Label 资源应用

实验环境：
本章节 Kubernetes 集群环境如下：

角色	IP	主机名	组件	硬件
控制节点	192.168.128.11	k8s-master01	apiserver controller-manager scheduler etcd containerd	CPU：4vCPU 硬盘：100G 内存：4GB 开启虚拟化
工作节点	192.168.128.21	k8s-node01	kubelet kube-proxy containerd calico coredns	CPU：6vCPU 硬盘：100G 内存：8GB 开启虚拟化
工作节点	192.168.128.22	k8s-node02	kubelet kube-proxy containerd calico coredns	CPU：6vCPU 硬盘：100G 内存：8GB 开启虚拟化

张岩峰老师微信，加我微信，邀请你加入 VIP 交流答疑群：
微信号：ZhangYanFeng0429
二维码：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**



1、Namespace（命名空间）

1.1、什么是命名空间？

Namespace（命名空间）是 Kubernetes 系统中的另一个非常重要的概念，Namespace 在很多情况下用于实现多租户的资源隔离。Namespace 通过将集群内部的资源对象“分配”到不同的 Namespace 中，形成逻辑上分组的不同项目、小组或用户组，便于不同的分组在共享使用整个集群的资源的同时还能被分别管理。

Kubernetes 支持多个虚拟集群，它们底层依赖于同一个物理集群。这些虚拟集群被称为命名空间。

命名空间 namespace 是 k8s 集群级别的资源，可以给不同的用户、租户、环境或项目创建对应的命名空间，例如，可以为 test、development、production 环境分别创建各自的命名空间。

1.2、NameSpace 应用场景

Kubernetes Namespace 可以将不同的资源隔离到不同的虚拟集群中，因此适用于以下应用场景：

1、多租户部署：不同的团队或租户可以在同一个 Kubernetes 集群中创建自己的 Namespace，从而实现资源隔离和权限隔离。

2、多环境部署：在同一个 Kubernetes 集群中，可以为不同的环境（如开发、测试、生产等）创建不同的 Namespace，从而避免资源冲突和环境混乱。

3、应用拆部署：将一个大型应用拆分为多个微服务，每个微服务可以通过创建独立的 Namespace 来进行资源隔离和管理。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

4、测试与运维：测试人员可以在自己的 Namespace 中部署测试环境，运维人员可以在自己的 Namespace 中进行运维操作。

5、安全和合规：通过为不同的应用程序或服务创建独立的 Namespace，可以增强安全性和合规性，从而避免可能的安全漏洞或数据泄露。

总之，Kubernetes Namespace 可以灵活地将不同的资源隔离到不同的虚拟集群中，从而实现资源隔离、权限隔离、环境隔离等多种应用场景，提高了应用程序的可维护性、可伸缩性和安全性。

1.3、NameSpace 实现原理

Kubernetes Namespace 是 Kubernetes 中一种资源隔离机制，用于将不同的资源按照所属团队、环境、用途等进行划分，从而避免出现资源冲突和安全问题。

Kubernetes Namespace 实现的原理如下：

1、每个 Namespace 都是 Kubernetes 集群中的一个虚拟集群，其内部包含的资源（Pod、Service、Volume、Secret 等）只能在该 Namespace 中使用。

2、Kubernetes API Server 使用 etcd 存储资源对象的元数据，每个 Namespace 都有一个独立的 etcd 存储空间，用于存储该 Namespace 中所有资源对象的元数据。

3、在 Kubernetes 运行时，每个 Namespace 的 Pod、Service、Volume、Secret 等资源都被分配一个唯一标识符（UID），并与该 Namespace 的标识符进行关联。这样，即使多个 Namespace 中的资源对象名称相同，它们也不会冲突。

4、Kubernetes Scheduler 根据不同的 Namespace 分别调度 Pod 到不同的 Node 上。

5、Kubernetes DNS 插件会为每个 Namespace 分别创建一个 DNS 域名，用于解析该 Namespace 中所有 Service 的域名。

总之，Kubernetes Namespace 实现的本质是通过为不同的资源对象分配唯一的标识符，从而将它们隔离在不同的虚拟集群中，并通过 Kubernetes API Server 和 etcd 存储元数据，并使用 Kubernetes Scheduler 和 DNS 插件进行资源调度和解析。

1.4、NameSpace 使用案例

● 1、查看名称空间及其资源对象

k8s 集群默认提供了几个名称空间用于特定目的，例如，kube-system 主要用于运行系统级资源，存放 k8s 一些组件的。而 default 则为那些未指定名称空间的资源操作提供一个默认值。

使用 `kubectl get namespace` 可以查看 namespace 资源，使用 `kubectl describe namespace $NAME` 可以查看特定的名称空间的详细信息。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

Kubernetes 集群在启动后会创建一个名为 default 的 Namespace，通过 kubectl 可以查看：

```
[root@k8s-master01 ~]# kubectl get namespaces
NAME                STATUS    AGE
calico-apiserver    Active    38h
calico-system        Active    38h
default              Active    38h
kube-node-lease      Active    38h
kube-public          Active    38h
kube-system          Active    38h
```

● 2、管理 namespace 资源

namespace 资源属性较少，通常只需要指定名称即可创建，如“kubectl create namespace devops”。namespace 资源的名称仅能由字母、数字、下划线、连接线等字符组成。删除 namespace 资源会级联删除其包含的所有其他资源对象。

● 应用案例

实例 1：创建一个 devops 命名空间

```
[root@k8s-master01 ~]# kubectl create namespace devops
namespace/devops created

或使用如下缩写也可创建：
[root@k8s-master01 ~]# kubectl create ns devops
```

实例 2：删除 devops 命名空间

```
[root@k8s-master01 ~]# kubectl delete ns devops
namespace "devops" deleted
```

实例 3：切换命名空间

```
# 切换到 kube-system 命名空间下
[root@k8s-master01 ~]# kubectl config set-context --current
--namespace=kube-system
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
[root@k8s-master01 ~]# kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
resource-nginx 1/1     Running   0           34m
```

← 当前在default命名空间下（默认）
← 切换到kube-system命名空间下

```
[root@k8s-master01 ~]# kubectl config set-context --current --namespace=kube-system
Context "kubernetes-admin@kubernetes" modified.
[root@k8s-master01 ~]# kubectl get pods
NAME                                                    READY   STATUS    RESTARTS   AGE
calico-kube-controllers-7b8458594b-gj5hj              1/1     Running   0           18h
calico-node-5t8qd                                       1/1     Running   0           18h
calico-node-7k9w2                                       1/1     Running   0           18h
calico-node-wt2h5                                       1/1     Running   1 (57m ago)  18h
coredns-6d8c4cb4d-2qskw                                1/1     Running   1 (57m ago)  18h
coredns-6d8c4cb4d-cm74s                                1/1     Running   1 (57m ago)  18h
etcd-k8s-master01                                      1/1     Running   6 (57m ago)  18h
kube-apiserver-k8s-master01                            1/1     Running   5 (57m ago)  18h
kube-controller-manager-k8s-master01                  1/1     Running   4 (57m ago)  18h
kube-proxy-6295r                                       1/1     Running   0           18h
kube-proxy-t7szc                                       1/1     Running   0           18h
kube-proxy-xlg7t                                       1/1     Running   2 (57m ago)  18h
kube-scheduler-k8s-master01                           1/1     Running   3 (57m ago)  18h
```

再次查看资源，默认就是 kube-system 下的资源了。

参数说明：
set-context: 在 kubeconfig 中设置上下文项
--current: 修改当前上下文
--namespace=kube-system: 修改当前命名空间为 kube-system

实例 4：查看哪些资源属于命名空间级别的

```
[root@k8s-master01 ~]# kubectl api-resources --namespaced=true
```

```
[root@k8s-master01 ~]# kubectl api-resources --namespaced=true
NAME          SHORTNAMES  APIVERSION  NAMESPACED  KIND
bindings      cm          v1          true        Binding
configmaps    cm          v1          true        ConfigMap
endpoints     ep          v1          true        Endpoints
events        ev          v1          true        Event
limitranges   limits      v1          true        LimitRange
persistentvolumeclaims pvc        v1          true        PersistentVolumeClaim
pods          po          v1          true        Pod
podtemplates  podtemplates v1          true        PodTemplate
replicationcontrollers rc          v1          true        ReplicationController
resourcequotas quota        v1          true        ResourceQuota
secrets       sa          v1          true        Secret
serviceaccounts sa          v1          true        ServiceAccount
services      svc          v1          true        Service
controllerrevisions cr          v1          true        ControllerRevision
daemonsets    ds          apps/v1     true        DaemonSet
deployments   deploy      apps/v1     true        Deployment
replicasets   rs          apps/v1     true        ReplicaSet
statefulsets  sts          apps/v1     true        StatefulSet
localsubjectaccessreviews localsubjectaccessreviews v1          true        LocalSubjectAccessReview
horizontalpodautoscalers hpa         autoscaling/v1 true        HorizontalPodAutoscaler
cronjobs      cj          batch/v1beta1 true        CronJob
jobs          batch/v1beta1 true        Job
leases        coordination.k8s.io/v1 true        Lease
networkpolicies networkpolicy.k8s.io/v1 true        NetworkPolicy
networksets   crd.projectcalico.org/v1 true        NetworkSet
endpointslices discovery.k8s.io/v1beta1 true        EndpointSlice
events        events.k8s.io/v1 true        Event
ingresses     ing          extensions/v1beta1 true        Ingress
ingresses     ing          networking.k8s.io/v1 true        Ingress
networkpolicies netpol      networking.k8s.io/v1 true        NetworkPolicy
poddisruptionbudgets pdb         policy/v1beta1 true        PodDisruptionBudget
rolebindings  rbac.authorization.k8s.io/v1 true        RoleBinding
roles         rbac.authorization.k8s.io/v1 true        Role
```

“kubectl api-resources” 这个命令可以查看 k8s 集群默认支持的所有资源。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

1.5、NameSpace 资源限额

1.5.1、资源限额介绍

在 Kubernetes 中, Namespace 可以用来组织和隔离应用资源, 而在 Namespace 中, 可以为应用设置资源配额。资源配额可以限制 Namespace 中某些资源的使用量, 包括 CPU、内存和存储等。下面是一些常见的 Namespace 资源配额:

- 1、CPU 配额: 可以限制一个 Namespace 中所有 Pod 使用的 CPU 配额。
- 2、内存配额: 可以限制一个 Namespace 中所有 Pod 使用的内存配额。
- 3、存储配额: 可以限制一个 Namespace 中所有 PersistentVolumeClaim(PVC) 使用的存储配额。
- 4、Pod 配额: 可以限制一个 Namespace 中最多可以运行的 Pod 数量。
- 5、服务配额: 可以限制一个 Namespace 中最多可以创建的 Service 数量。

这些资源配额可以通过 Kubernetes 中的资源配额 (ResourceQuotas) API 对象来定义和应用。当应用超过指定资源配额时, Kubernetes 将会阻止容器创建, 并发送警告消息。通过在 Namespaces 中使用资源配额, 管理员可以控制资源使用, 确保应用程序不会消耗整个 Kubernetes 环境中的资源。

注意:

- 1、在集群容量小于各命名空间配额总和的情况下, 可能存在资源竞争。资源竞争时, Kubernetes 系统会遵循先到先得的原则。
- 2、不管是资源竞争还是配额的修改, 都不会影响已经创建的资源使用对象。

1.5.2、启用资源配额

资源配额的支持在很多 Kubernetes 版本中是默认启用的。当 API 服务器的命令行标志 “--enable-admission-plugins=” 中包含 “ResourceQuota” 时, 资源配额会被启用。

检查当前集群是否启用了资源配额:

```
[root@k8s-master01 ~]# cat /etc/kubernetes/manifests/kube-apiserver.yaml |  
grep "enable-admission-plugins"  
- --enable-admission-plugins=NodeRestriction
```

这里的 “--enable-admission-plugins=” 参数没有指定 “ResourceQuota”, 但是可以使用资源限额, 在当前集群版本中默认是启用的。我们也可以再给他加上去。

```
[root@k8s-master01 ~]# vi /etc/kubernetes/manifests/kube-apiserver.yaml  
- --enable-admission-plugins=NodeRestriction, ResourceQuota  
[root@k8s-master01 ~]# systemctl restart kubelet
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

1.5.3、计算资源配额

下面进行对命名空间下的可被请求的“计算资源”总量进行限制。如：“CPU、内存”。

● 配额机制所支持的资源类型

资源名称	描述
limits.cpu	所有非终止状态的 Pod，其 CPU 限额总量不能超过该值。
limits.memory	所有非终止状态的 Pod，其内存限额总量不能超过该值。
requests.cpu	所有非终止状态的 Pod，其 CPU 需求总量不能超过该值。
requests.memory	所有非终止状态的 Pod，其内存需求总量不能超过该值。
cpu	与 requests.cpu 相同
memory	与 requests.memory 相同

● 项目演练

目标：创建 devops 命名空间，对 devops 命名空间进行资源配额，限制 CPU 总量不得超过 4 核、内存总量不得超过 4G；限制请求 CPU 大小不得超过 2 核、内存大小不得超过 2G。

(1) 创建命名空间

必须先将命名空间创建出来，才能够对已存在的命名空间进行资源限额。

```
[root@k8s-master01 ~]# kubectl create ns devops
namespace/devops created
```

(2) 创建 yaml 文件，对命名空间进行限额

```
[root@k8s-master01 ~]# vi devops-quota.yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: mem-cpu-quota
  namespace: devops
spec:
  hard:
    requests.cpu: "2"
    requests.memory: 2Gi
    limits.cpu: "4"
    limits.memory: 4Gi
```

对上面的 yaml 文件解释如下：

创建的 ResourceQuota 对象将在 devops 名字空间中添加以下限制：

每个容器必须设置内存请求 (memory request)，内存限额 (memory limit)，cpu 请求 (cpu request) 和 cpu 限额 (cpu limit)。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

- requests.cpu: "2" 所有容器的CPU请求总额不得超过2CPU。
- requests.memory: 2Gi 所有容器的内存请求总额不得超过2GiB。
- limits.cpu: "4" 所有容器的CPU限额总额不得超过4CPU。
- limits.memory: 4Gi 所有容器的内存限额总额不得超过4GiB。

(3) 更新清单文件

```
[root@k8s-master01 ~]# kubectl apply -f devops-quota.yaml
resourcequota/mem-cpu-quota created

[root@k8s-master01 ~]# kubectl get resourcequota -A
[root@k8s-master01 ~]# kubectl get resourcequota -A
NAMESPACE  NAME          AGE  REQUEST                                     LIMIT
devops     mem-cpu-quota 2m9s requests.cpu: 0/2, requests.memory: 0/2Gi limits.cpu: 0/4, limits.memory: 0/4Gi
[root@k8s-master01 ~]#
```

(4) 检查命名空间限额

```
[root@k8s-master01 ~]# kubectl describe ns devops
Name:          devops
Labels:        kubernetes.io/metadata.name=devops
Annotations:   <none>
Status:        Active

Resource Quotas
  Name:          mem-cpu-quota
  Resource       Used  Hard
  -----
  limits.cpu     0    4
  limits.memory  0    4Gi
  requests.cpu   0    2
  requests.memory 0    2Gi

No LimitRange resource.
```

(5) 创建测试 pod 容器

注意：创建 pod 时必须设置资源限额，否则创建失败。

```
[root@k8s-master01 ~]# vi pod-test.yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-test
  namespace: devops
  labels:
    app: tomcat-pod-test
spec:
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
containers:
- name: tomcat-test
  ports:
  - containerPort: 8080
  image: tomcat
  resources:
    limits:
      memory: "2Gi"
      cpu: "1"
    requests:
      memory: "500Mi"
      cpu: "500m"
```

(6) 创建 Pod，并检查

```
# 创建 pod
[root@k8s-master01 yaml]# kubectl create -f pod-test.yaml
pod/pod-test created

# 查看 pod 是否启动
[root@k8s-master01 yaml]# kubectl get pods -n devops
NAME          READY   STATUS    RESTARTS   AGE
pod-test      1/1     Running   0           5s

# 查看命名空间资源限额情况
[root@k8s-master01 ~]# kubectl describe ns devops
Name:          devops
Labels:         kubernetes.io/metadata.name=devops
Annotations:    <none>
Status:        Active

Resource Quotas
Name:          mem-cpu-quota
Resource       Used    Hard
-----
limits.cpu     1      4
limits.memory  2Gi    4Gi
requests.cpu   500m   2
requests.memory 500Mi  2Gi

No LimitRange resource.
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

1.5.4、存储资源配额

下面进行对命名空间下的可被请求的“存储资源”总量进行限制。如：“创建 PVC 的数量、PVC 的总大小”。

● 配额机制所支持的资源类型

资源名称	描述
requests.storage	所有 PVC，存储资源的需求总量不能超过该值。
persistentvolumeclaims	在该命名空间中所允许的 PVC 总量。
<storage-class-name>.storageclass.storage.k8s.io/requests.storage	在所有与<storage-class-name>相关的持久卷申领中，存储请求的总和不能超过该值。
<storage-class-name>.storageclass.storage.k8s.io/persistentvolumeclaims	在与 storage-class-name 相关的所有持久卷申领中，命名空间中可以存在的持久卷申领总数。

● 项目演练

目标：创建 storage 命名空间，对 storage 命名空间进行资源配额，限制 PVC 资源存储总量不能超过 50G，PVC 总量不得超过 1。

(1) 创建命名空间

```
[root@k8s-master01 ~]# kubectl create ns storage
namespace/storage created
```

(2) 创建 yaml 文件，对命名空间进行限额

```
[root@k8s-master01 ~]# vi storage-quota.yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: pvc-quota
  namespace: storage
spec:
  hard:
    requests.storage: "50Gi"
    persistentvolumeclaims: 1

[root@k8s-master01 ~]# kubectl apply -f storage-quota.yaml
resourcequota/pvc-quota created
```

对上面的 yaml 文件解释如下：

创建的 ResourceQuota 对象将在 devops 名字空间中添加以下限制：

- requests.storage：存储资源的总量不得超过 50G。
- persistentvolumeclaims：pvc 的数量不得超过 1。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

(3) 检查命名空间限额

```
[root@k8s-master01 ~]# kubectl get ResourceQuota -n storage
```

NAME	AGE	REQUEST	LIMIT
pvc-quota	81s	persistentvolumeclaims: 0/1, requests.storage: 0/50Gi	

```
[root@k8s-master01 ~]#
```



```
[root@k8s-master01 ~]# kubectl describe ns storage
```

Name: storage
Labels: kubernetes.io/metadata.name=storage
Annotations: <none>
Status: Active

Resource Quotas

Name:	pvc-quota	
Resource	Used	Hard
-----	---	---
persistentvolumeclaims	0	1
requests.storage	0	50Gi

No LimitRange resource.

(4) 创建测试 pvc，大于 50G

```
[root@k8s-master01 ~]# vi storage-pvc.yaml
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: storage-pvc
  namespace: storage
spec:
  accessModes: ["ReadWriteMany"]
  resources:
    requests:
      storage: 60Gi
```



```
[root@k8s-master01 ~]# kubectl apply -f storage-pvc.yaml
```

Error from server (Forbidden): error when creating "storage-pvc.yaml": persistentvolumeclaims "storage-pvc" is forbidden: exceeded quota: pvc-quota, requested: requests.storage=60Gi, used: requests.storage=0, limited: requests.storage=50Gi

这个错误意味着你创建的 PVC 对应的 Namespace 已经达到了配额限制，容量

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

不足，无法再创建指定容量的 PVC。具体来说，这个 PVC 要求的存储资源大小是 60GB，但是当前 Namespace 配额限制中设置的存储配额大小限制是 50GB，因此 Kubernetes 无法创建这个 PVC。

要解决这个问题，需要修改 Namespace 的配额限制，或者先删除一些不必要的 PVC 或文件以释放空间。

(5) 创建测试 pvc，小于 50G

```
[root@k8s-master01 ~]# vi storage-pvc.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: storage-pvc
  namespace: storage
spec:
  accessModes: ["ReadWriteMany"]
  resources:
    requests:
      storage: 40Gi

[root@k8s-master01 ~]# kubectl apply -f storage-pvc.yaml
persistentvolumeclaim/storage-pvc created
```

(6) 查看命名空间资源限额情况

```
[root@k8s-master01 ~]# kubectl get ResourceQuota -n storage
```

NAME	AGE	REQUEST	LIMIT
pvc-quota	10m	persistentvolumeclaims: 1/1, requests.storage: 40Gi/50Gi	

```
[root@k8s-master01 ~]#
```

```
[root@k8s-master01 ~]# kubectl describe ns storage
```

```
Name:          storage
Labels:        kubernetes.io/metadata.name=storage
Annotations:   <none>
Status:        Active
```

```
Resource Quotas
```

Name:	pvc-quota	
Resource	Used	Hard
-----	---	---
persistentvolumeclaims	1	1
requests.storage	40Gi	50Gi

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
No LimitRange resource.
```

1.5.5、对象数量配额

下面进行对命名空间下的可被请求的“资源”总量进行限制。

● 配额机制所支持的资源类型

资源名称	描述
configmaps	在该命名空间中允许存在的 ConfigMap 总数上限。
persistentvolumeclaims	在该命名空间中允许存在的 PVC 的总数上限。
Pods	在该命名空间中允许存在的非终止状态的 Pod 总数上限。
replicationcontrollers	在该命名空间中允许存在的 ReplicationController 总数上限。
resourcequotas	在该命名空间中允许存在的 ResourceQuota 总数上限。
services	在该命名空间中允许存在的 Service 总数上限。
services.loadbalancers	在该命名空间中允许存在的 LoadBalancer 类型的 Service 总数上限。
services.nodeports	在该命名空间中允许存在的 NodePort 类型的 Service 总数上限。
secrets	在该命名空间中允许存在的 Secret 总数上限。

● 项目演练

目标：创建 number 命名空间，对 number 命名空间进行资源配额，限制 configmaps 资源总量不能超过 10、PVC 资源总量不得超过 5、pods 资源总量不得超过 50、services 资源总量不得超过 10、services.nodeports 资源总量不得超过 5、secrets 资源总量不得超过 10。

(1) 创建命名空间

```
[root@k8s-master01 ~]# kubectl create ns number
namespace/number created
```

(2) 创建 yaml 文件，对命名空间进行限额

```
[root@k8s-master01 ~]# vi number-quota.yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: number-quota
  namespace: number
spec:
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
hard:
  configmaps: 10
  persistentvolumeclaims: 5
  pods: 50
  services: 10
  services.nodeports: 5
  secrets: 10

[root@k8s-master01 ~]# kubectl apply -f number-quota.yaml
resourcequota/number-quota created
```

(3) 检查命名空间限额

```
[root@k8s-master01 ~]# kubectl get ResourceQuota -n number
[root@k8s-master01 ~]# kubectl get ResourceQuota -n number
NAME      AGE  REQUEST
number-quota 25s  configmaps: 1/10, persistentvolumeclaims: 0/5, pods: 0/50, secrets: 0/10, services: 0/10, services.nodeports: 0/5
[root@k8s-master01 ~]#

[root@k8s-master01 ~]# kubectl describe ns number
Name:      number
Labels:     kubernetes.io/metadata.name=number
Annotations: <none>
Status:     Active

Resource Quotas
  Name:      number-quota
  Resource   Used  Hard
  -----
  configmaps 1    10
  persistentvolumeclaims 0    5
  pods        0    50
  secrets     0    10
  services    0    10
  services.nodeports 0    5

No LimitRange resource.
```

2、Label（标签）

2.1、什么是标签？

Label（标签）是 Kubernetes 系统中另外一个核心概念。一个 Label 是一个 key=value 的键值对，其中 key 与 value 由用户自己指定。Label 可以被附加到各种资源对象上，例如 Node、Pod、Service、RC 等，在 k8s 中，大部分资源都

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

可以打标签。一个资源对象可以定义任意数量的 Label，同一个 Label 也可以被添加到任意数量的资源对象上。Label 通常在资源对象定义时确定，也可以在对象创建后动态添加或者删除。

我们可以通过给指定的资源对象捆绑一个或多个不同的 Label 来实现多维度的资源分组管理功能，以便灵活、方便地进行资源分配、调度、配置、部署等管理工作。例如，部署不同版本的应用到不同的环境中；监控和分析应用（日志记录、监控、告警）等。

在前面我们已经说过了标签的概念，这里我们主要来讲解在 K8S 集群中标签的操作。

2.2、给 pod 资源打标签

● 语法

```
# 创建标签语法
kubectl label 资源类型 资源名称 <label-key>=<label-value>

# 删除标签语法
kubectl label 资源类型 资源名称 <label-key>-

# 查看标签语法
kubectl get 资源类型 {资源名称} --show-labels
```

● 应用案例

创建测试容器：

```
[root@k8s-master01 ~]# kubectl run nginx --image=nginx
pod/nginx created

[root@k8s-master01 ~]# kubectl get pods
NAME      READY   STATUS    RESTARTS   AGE
nginx     1/1     Running   0           45s
```

实例 1：查看 pod 的标签

```
[root@k8s-master01 ~]# kubectl get pods --show-labels
NAME      READY   STATUS    RESTARTS   AGE   LABELS
nginx     1/1     Running   0           84s   run=nginx
```

实例 2：给已存在的 Pod 打标签

```
[root@k8s-master01 ~]# kubectl label pods nginx release=v1 app=web
pod/nginx labeled
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
[root@k8s-master01 ~]# kubectl get pods --show-labels
```

NAME	READY	STATUS	RESTARTS	AGE	LABELS
nginx	1/1	Running	0	3m2s	app=web,release=v1,run=nginx

```
[root@k8s-master01 ~]#
```

实例 3: 删除 Pod 资源标签

```
[root@k8s-master01 ~]# kubectl label pods nginx release-  
pod/nginx unlabeled
```

```
[root@k8s-master01 ~]# kubectl get pods --show-labels
```

NAME	READY	STATUS	RESTARTS	AGE	LABELS
nginx	1/1	Running	0	4m	app=web,run=nginx

2.3、查看资源标签

● 应用案例

实例 1: 查看默认名称空间下所有 pod 资源的标签

```
[root@k8s-master01 ~]# kubectl get pods --show-labels
```

实例 2: 查看默认名称空间下指定 pod 具有的所有标签

```
[root@k8s-master01 ~]# kubectl get pods nginx --show-labels
```

实例 3: 列出默认名称空间下标签 key 是 app 的 pod，不显示标签

```
[root@k8s-master01 ~]# kubectl get pods -l app
```

显示标签加--show-labels

```
[root@k8s-master01 ~]# kubectl get pods -l app --show-labels
```

实例 4: 列出默认名称空间下标签 key 是 app、值是 web 的 pod，不显示标签

```
[root@k8s-master01 ~]# kubectl get pods -l name=tomcat
```

实例 5: 列出默认名称空间下标签 key 是 name 的所有 pod，并打印对应的标签值

```
[root@k8s-master01 ~]# kubectl get pods -L name
```

实例 6: 查看所有名称空间下的所有 pod 的标签

```
[root@k8s-master01 ~]# kubectl get pods --all-namespaces  
--show-labels
```

实例 7: 列出默认名称空间下标签 key 是 app 的 pod，并显示 key 的值

```
[root@k8s-master01 ~]# kubectl get pods -L app
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

NAME	READY	STATUS	RESTARTS	AGE	APP
nginx	1/1	Running	0	8m53s	web

2.4、通过标签选择器删除 pod

1、查看 pod 中的标签

```
[root@k8s-master01 ~]# kubectl get pod --show-labels
NAME          READY    STATUS    RESTARTS   AGE    APP
nginx         1/1     Running   0           8m53s  web
[root@k8s-master01 ~]# kubectl get pod --show-labels
NAME          READY    STATUS    RESTARTS   AGE    LABELS
nginx         1/1     Running   0           54m    app=web,run=nginx
[root@k8s-master01 ~]#
```

2、删除带有 app=web 标签的 pod

```
[root@k8s-master01 ~]# kubectl delete pods -l app=web
pod "nginx" deleted
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**