

## 31 资源隔离：Namespace 剖析

更新时间：2020-10-19 09:54:02



读一本好书，就是和许多高尚的人谈话。——歌德

在介绍 `namespace` 之前先举个大数据的例子，很多大公司为了管理方便都有一个规模非常大的集群，可能有几千上万台机器。同时在公司内部又存在不同的部门，不同的业务，为了区分开各个部门的集群使用情况，同时也为了避免业务之间互相影响，一般都会将集群资源划分出多个资源队列，不同的业务使用不同的资源队列。

类似的，`Kubernetes` 集群里面也有一个类似资源队列的角色，叫 `namespace`。在生产实践中我们可以为不同的部门设置不同的 `namespace`，另外 `namespace` 支持资源限制，我们可以按需为不同的部门设置不同的资源限制。

### 1. 背景

正如前面所说，`namespace` 适用于跨团队、跨项目的多用户使用场景。如果 `Kubernetes` 集群的使用者很少，那么不需要考虑 `namespace`。

`Namespace` 为资源提供了一个界限，同一个 `namespace` 内的资源必须保证名字唯一，不同 `namespace` 内的资源可以名字相同。这里的资源包括 `Pod`, `Deployment`, `Service` 等。`namespace` 不支持嵌套结构，即一个 `namespace` 下包含子 `namespace`。这里值得注意的是：命名空间资源本身不受限于一个 `namespace` 内，像一些底层资源，比如 `nodes` 和持久化卷也不属于任何 `namespace`。

`Namespace` 可以通过 `resource quota` 设置资源配额，这样我们就可以为不同用户群体划分资源了。

在 `Kubernetes` 的未来版本，同一个 `namespace` 下的对象都将使用相同的访问控制。

值得注意的是，不要滥用 `namespace`，比如不要因为资源存在微小的差异（比如版本不同），就将这些资源划分到不同的 `namespace`。对于这种情况，可以使用 `label` 去区分。

## 2. Namespace 使用

### 查看 namespace

和其他 `kubernetes` 的资源一样，我们可以通过命令 `kubectl get namespace` 或者 `kubectl get ns` 来获取集群中的 `namespace` 情况，下面是我自己的 `kubernetes` 集群中的 `namespace` 情况。

```
~ $ kubectl get namespace
NAME      STATUS  AGE
default   Active  148d
imooc     Active  57m
kube-node-lease  Active  148d
kube-public  Active  148d
kube-system  Active  148d
```

这其中有一个资源是由 `kubernetes` 自动创建的：

- **default**: 创建对象时没有指定特定 `namespace` 的对象都会被置于 `default` `namespace` 下。
- **kube-system**: `Kubernetes` 系统创建对象所使用的命名空间，比如 `kube-proxy`、`kube-scheduler` 等。
- **kube-public**: 这个命名空间是自动创建的，所有用户（包括未经过身份验证的用户）都可以读取它。这个命名空间主要用于集群使用，以防某些资源在整个集群中应该是可见和可读的。这个命名空间的公共方面只是一种约定，而不是要求。

### 为请求设置 namespace

这里说的请求，包括创建、获取、删除特定的资源对象等，可以通过 `-namespace=<namespace name>` 或者 `-n <namespace name>` 来指定特定的 `namespace`。

```
kubectl run nginx --image=nginx --namespace=<insert-namespace-name-here>
kubectl get pods --namespace=<insert-namespace-name-here>
```

### 设置 namespace 首选项

如果觉得每个 `kubectl` 命令后面都添加 `--namespace` 参数过于繁琐，则可以通过 `kubectl config` 为当前 `session` 设置 `namespace` 参数，则之后的 `kubectl` 命令都将使用这个特定的 `namespace`。

```
kubectl config set-context --current --namespace=<insert-namespace-name-here>
# Validate it
kubectl config view | grep namespace:
```

## Namespace 和 DNS

当我们创建一个 `Service` 对象时，`Kubernetes` 为了在集群内部可以访问到该 `Service` 对象的域名，会自动创建一个 `DNS` 条目，形式为 `<service-name>.<namespace-name>.svc.cluster.local`，也就是说如果容器只使用 `<service-name>`，它将被解析到本地命名空间的服务。这对于跨多个命名空间（如开发、分级和生产）使用相同的配置非常有用。如果您希望跨命名空间访问，则需要使用完全限定域名（`FQDN`）。

## 3. 为 Namespace 设置资源配额

Namespace 机制的一个目标就是通过 `namespace` 将集群资源划分给不同的业务方使用，这一小节我们就来看一下如何限制一个 `namespace` 的资源配额。

在 Kubernetes 中，资源配额是通过一个叫 `ResourceQuota` 的对象来定义的。通过 `ResourceQuota` 我们可以限制 `namespace` 中的对象数量，以及可以使用的计算资源总量。

Resource Quota 默认是支持的，如果发现你的 `kubernetes` 不支持的话，可以检查参数 `--enable-admission-plugins` 的值里面有没有 `ResourceQuota`。

`ResourceQuota` 提供的资源配额支持包括：

## 计算资源配置

Resource	Description
<code>limits.memory</code>	所有非终结态的 Pod 的 <code>memory limits</code> 的和不能超过这个值
<code>limits.cpu</code>	所有非终结态的 Pod 的 <code>cpu limits</code> 的和不能超过这个值
<code>requests.cpu</code>	所有非终结态的 Pod 的 <code>cpu requests</code> 的和不能超过这个值
<code>requests.memory</code>	所有非终结态的 Pod 的 <code>memory requests</code> 的和不能超过这个值

## 存储资源配置

Resource	Description
<code>Requests.storage</code>	所有 PVC 的存储请求总和不能超过这个值
<code>persistentvolumeclaims</code>	<code>namespace</code> 中的 pvc 数量
<code>&lt;storage-class-name&gt;.storageclass.storage.k8s.io/requests.storage</code>	指定 <code>storage-class-name</code> 的所有 pvc 的存储请求的上限值
<code>&lt;storage-class-name&gt;.storageclass.storage.k8s.io/persistentvolumeclaims</code>	指定 <code>storage-class-name</code> 的 pvc 数量

## 对象数量配额

Name	Description
<code>configmaps</code>	<code>namespace</code> 中可以存在的配置映射的总数。
<code>persistentvolumeclaims</code>	<code>namespace</code> 中可以存在的 PVC 总数。
<code>pods</code>	<code>namespace</code> 中可以存在的非终止态的 pod 总数。如果一个 pod 的 <code>status.phase</code> 是 <code>Failed, Succeeded</code> ，则该 pod 处于终止态。
<code>replicationcontrollers</code>	<code>namespace</code> 中可以存在的 rc 总数。
<code>resourcequotas</code>	<code>namespace</code> 中可以存在的资源配额（ <code>resource quotas</code> ）总数。
<code>services</code>	<code>namespace</code> 中可以存在的服务总数量。
<code>services.loadbalancers</code>	<code>namespace</code> 中可以存在的服务的负载均衡的总数量。
<code>services.nodeports</code>	<code>namespace</code> 中可以存在的服务的主机接口的总数量。
<code>secrets</code>	<code>namespace</code> 中可以存在的 <code>secrets</code> 的总数量。

我们下面来演示一下 `Resource Quota` 如何使用。首先定义个 `resource quota` 的资源文件，我们这里使用一个 `List` 对象，下面可以连接多个 `Resource Quota` 对象。

```
apiVersion: v1
kind: List
items:
- apiVersion: v1
  kind: ResourceQuota
  metadata:
    name: quota
  spec:
    hard:
      configmaps: "20"
      limits.cpu: "4"
      limits.memory: 10Gi
      persistentvolumeclaims: "10"
      pods: "30"
      requests.storage: 10Ti
      secrets: "60"
      services: "40"
      services.loadbalancers: "50"
```

然后使用 `kubectl apply` 将这个 `ResourceQuota` 对象应用到指定的 `namespace` 中。

```
□ kubectl apply -f resourcequota.yaml -n imooc
```

最后我们再通过 `kubectl describe` 看一下这个 `namespace`, 可以看到其中的 `Resource Quotas` 部分。

```
□ k8s kubectl describe ns imooc
Name:      imooc
Labels:    <none>
Annotations: <none>
Status:    Active

Resource Quotas
Name:      quota
Resource      Used  Hard
-----
configmaps      0    20
limits.cpu      700m  4
limits.memory    900Mi 10Gi
persistentvolumeclaims 0    10
pods          1    30
requests.storage  0    10Ti
secrets        1    60
services       0    40
services.loadbalancers 0    50

Resource Limits
Type      Resource  Min  Max  Default Request  Default Limit  Max Limit/Request Ratio
-----
Container  memory   99Mi  1Gi  111Mi      900Mi      -
Container  cpu     100m  800m 110m      700m      -
```

## 4. 为 `Namespace` 设置资源限制

在 `Kubernetes` 集群中, 容器可以使用的资源默认没有上限。为了避免单个容器或者 `Pod` 用光 `node` 上的所有可用资源, `Kubernetes` 提供了一种可以给 `namespace` 设置资源限制的方式, 叫 `LimitRange`。`LimitRange` 提供的限制包括:

- 限制每个 `Pod` 或者容器可以使用的最少和最多计算资源
- 限制每个 `PVC` (`PersistentVolumeClaim`) 可以使用的最少和最多存储
- 限制资源的 `request` 和 `limit` 比例

- 设置容器默认的 `request/limit` 的资源大小

下面我们定一个一个 `LimitRange` 资源对象。

```
apiVersion: v1
kind: LimitRange
metadata:
  name: limit-mem-cpu-per-container
spec:
  limits:
  - max:
      cpu: "800m"
      memory: "1Gi"
    min:
      cpu: "100m"
      memory: "99Mi"
    default:
      cpu: "700m"
      memory: "900Mi"
    defaultRequest:
      cpu: "110m"
      memory: "111Mi"
  type: Container
```

然后我们将这个 `LimitRange` 对象应用到其中一个 `namespace` 中。

`kubectl apply -f limitrange.yaml -n imooc`

我们通过 `kubectl describe` 查看一下 `imooc` 这个 `namespace` 的一些描述信息。

```
k8s kubectl describe namespace imooc
Name:      imooc
Labels:    <none>
Annotations: <none>
Status:    Active

No resource quota.

Resource Limits
Type  Resource  Min  Max  Default Request  Default Limit  Max Limit/Request  Ratio
---  -----
Container  cpu    100m  800m  110m      700m          -
Container  memory  99Mi  1Gi   111Mi     900Mi         -
```

我们将 `namespace` 的描述信息和 `LimitRange` 资源对象联合起来一起看，可以得出一些结论：

- `spec.limits.type` 表示限制资源的类型，可以是 `Container` 或者 `Pod`
- `spec.limits.max` 表示计算资源限制的最大值，可以包含 `cpu` 或者 `memory`
- `spec.limits.min` 表示计算资源限制的最小值，可以包含 `cpu` 或者 `memory`
- `spec.limit.defaultRequest` 表示计算资源的默认 `request` 值，可以包含 `cpu` 或者 `memomry`
- `spec.limit.default` 表示计算资源的默认 `limit` 值，可以包含 `cpu` 或者 `memory`

为了更直观的感受 `LimitRange` 的作用，我们创建一个 `Pod` 看看（这里仅仅为了演示，生产实践中基本不用直接创建 `Pod` 对象）。

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: busybox
    command: ['sh', '-c', 'echo Hello Kubernetes! && sleep 3600']
```

还是通过 `kubectl apply` 这种声明式的 API 来创建。

```
kubectl apply -f busybox.pod.simple.yaml -n imooc
```

然后通过 `kubectl describe` 来查看 Pod 的资源情况，如下图所示，计算资源确实设置成 `LimitRange` 中预设的值了。

```
□ k8s kubectl describe pods myapp-pod -n imooc
Name:      myapp-pod
Namespace:  imooc
Priority:   0
...
Containers:
  myapp-container:
    ...
    Limits:
      cpu:  700m
      memory: 900Mi
    Requests:
      cpu:  110m
      memory: 111Mi
    ...
...
```

## 5. 总结

这篇文章介绍了 Kubernetes 中对资源进行隔离的方案：`namespace`，并介绍了如何给 `namespace` 设置资源限制等操作。希望对大家生产实践会有用。

```
}
```

← 30 Kubernetes 是什么？

32 Kubernetes 核心概念解析：  
Pod (→)