实践篇-准入控制

一、什么是K8S之准入控制

> 就是在创建资源经过身份验证之后，kube-apiserver在数据写入etcd之前做一次拦截，然后对资源进行更改、判断正确性等操作。

一个 *LimitRange（限制范围）* 对象提供的限制能够做到：

- 在一个命名空间中实施对每个 Pod 或 Container 最小和最大的资源使用量的限制。
- 在一个命名空间中实施对每个 PersistentVolumeClaim 能申请的最小和最大的存储空间大小的限制。
- 在一个命名空间中实施对一种资源的申请值和限制值的比值的控制。
- 设置一个命名空间中对计算资源的默认申请/限制值，并且自动的在运行时注入到多个 Container 中。

二、启用 LimitRang

- 对 LimitRange 的支持自 Kubernetes 1.10 版本默认启用。
- LimitRange 支持在很多 Kubernetes 发行版本中也是默认启用的。
- LimitRange 的名称必须是合法的 DNS 子域名。

三、限制范围总览

- 管理员在一个命名空间内创建一个 `LimitRange` 对象。
- 用户在命名空间内创建 Pod ，Container 和 PersistentVolumeClaim 等资源。
- `LimitRanger` 准入控制器对所有没有设置计算资源需求的 Pod 和 Container 设置默认值与限制值， 并跟踪其使用量以保证没有超出命名空间中存在的任意 LimitRange 对象中的最小、最大资源使用量以及使用量比值。
- 若创建或更新资源（Pod、 Container、PersistentVolumeClaim）违反了 LimitRange 的约束， 向 API 服务器的请求会失败，并返回 HTTP 状态码 `403 FORBIDDEN` 与描述哪一项约束被违反的消息。
- 若命名空间中的 LimitRange 启用了对 `cpu` 和 `memory` 的限制， 用户必须指定这些值的需求使用量与限制使用量。否则，系统将会拒绝创建 Pod。
- LimitRange 的验证仅在 Pod 准入阶段进行，不对正在运行的 Pod 进行验证。

能够使用限制范围创建的策略示例有：

- 在一个有两个节点，8 GiB 内存与16个核的集群中，限制一个命名空间的 Pod 申请 100m 单位，最大 500m 单位的 CPU，以及申请 200Mi，最大 600Mi 的内存。
- 为 spec 中没有 cpu 和内存需求值的 Container 定义默认 CPU 限制值与需求值 150m，内存默认需求值 300Mi。

在命名空间的总限制值小于 Pod 或 Container 的限制值的总和的情况下，可能会产生资源竞争。 在这种情况下，将不会创建 Container 或 Pod。

竞争和对 LimitRange 的改变都不会影响任何已经创建了的资源

四、配置LimitRange（对Pod进行配置、限额）

4.1、配置 CPU 最小和最大约束

```
# 创建一个命名空间，以便本练习中创建的资源和集群的其余资源相隔离
[root@k8s-master01 ~]# kubectl create namespace constraints-cpu-example
namespace/constraints-cpu-example created
# 1、LimitRange 的配置文件
[root@k8s-master01 ~]# cat cpu-constraints.yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: cpu-min-max-demo-lr
spec:
  limits:
  - max:
      cpu: "800m"
    min:
      cpu: "200m"
    type: Container

# 2、创建 LimitRange:
[root@k8s-master01 ~]# kubectl apply -f  cpu-constraints.yaml -n constraints-cpu-
example
limitrange/cpu-min-max-demo-lr created

# 3、查看结果
# 输出结果显示 CPU 的最小和最大限制符合预期。但需要注意的是，尽管你在 LimitRange 的配置文件中你没有声明默认值，默认值也会被自动创建。
[root@k8s-master01 ~]# kubectl get limitrange cpu-min-max-demo-lr --output=yaml --
namespace=constraints-cpu-example
# 只列出关键的
spec:
  limits:
  - default:
      cpu: 800m
    defaultRequest:
      cpu: 800m
    max:
      cpu: 800m
    min:
      cpu: 200m
    type: Container
```

现在不管什么时候在 constraints-cpu-example 命名空间中创建容器，Kubernetes 都会执行下面这些步骤：

- 如果容器没有声明自己的 CPU 请求和限制，将为容器指定默认 CPU 请求和限制。
- 核查容器声明的 CPU 请求确保其大于或者等于 200 millicpu。
- 核查容器声明的 CPU 限制确保其小于或者等于 800 millicpu。

流程就是说，我们创建了一个namespace=constraints-cpu-example的名称空间，然后在这个名称空间创建了一个LimitRange。然后现在不管什么时候在 constraints-cpu-example 命名空间中创建容器，Kubernetes 都会执行上面那些步骤！我们创建的LimitRange对这个namespace=constraints-cpu-example的名称空间里面起低调Pod都起了限制的作用

4.2、配置命名空间的最小和最大内存约束

LimitRange 的配置文件

```
apiVersion: v1
kind: LimitRange
metadata:
  name: mem-min-max-demo-lr
spec:
  limits:
  - max:
      memory: 1Gi
    min:
      memory: 500Mi
    type: Container
# 输出显示预期的最小和最大内存约束。 但请注意，即使你没有在 LimitRange 的配置文件中指定默认值，也会自动创建它们。
  limits:
  - default:
      memory: 1Gi
    defaultRequest:
      memory: 1Gi
    max:
      memory: 1Gi
    min:
      memory: 500Mi
    type: Container
```

现在，只要在 constraints-mem-example 命名空间中创建容器，Kubernetes 就会执行下面的步骤：

- 如果 Container 未指定自己的内存请求和限制，将为它指定默认的内存请求和限制。
- 验证 Container 的内存请求是否大于或等于500 MiB。
- 验证 Container 的内存限制是否小于或等于1 GiB。

4.3、配置 CPU 最小和最大约束

LimitRange 的配置文件：

```
apiVersion: v1
kind: LimitRange
metadata:
  name: cpu-min-max-demo-lr
spec:
  limits:
```

```
  - max:
      cpu: "800m"
    min:
      cpu: "200m"
    type: Container
#  输出结果显示 CPU 的最小和最大限制符合预期。但需要注意的是，尽管你在 LimitRange 的配置文件中你没有声明默认值，默认值也会被自动创建。
limits:
- default:
    cpu: 800m
  defaultRequest:
    cpu: 800m
  max:
    cpu: 800m
  min:
    cpu: 200m
  type: Container
```

现在不管什么时候在 constraints-cpu-example 命名空间中创建容器，Kubernetes 都会执行下面这些步骤：

- 如果容器没有声明自己的 CPU 请求和限制，将为容器指定默认 CPU 请求和限制。
- 核查容器声明的 CPU 请求确保其大于或者等于 200 millicpu。
- 核查容器声明的 CPU 限制确保其小于或者等于 800 millicpu。

说明： 当创建 LimitRange 对象时，你也可以声明大页面和 GPU 的限制。 当这些资源同时声明了 'default' 和 'defaultRequest' 参数时，两个参数值必须相同。

五、ResourceQuot（对名称空间进行配置、限额）

5.1、配置内存和 CPU 配额

```
# 1、创建命名空间，以便本练习中创建的资源和集群的其余部分相隔离。
[root@k8s-master01 ~]# kubectl create namespace quota-mem-cpu-example
namespace/quota-mem-cpu-example created

# 2、创建 ResourceQuota
[root@k8s-master01 ~]# vim quota-mem-cpu.yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: mem-cpu-demo
spec:
  hard:
    requests.cpu: "1"
    requests.memory: 1Gi
    limits.cpu: "2"
    limits.memory: 2Gi

# 3、创建 ResourceQuota
```

```
[root@k8s-master01 ~]# kubectl apply -f quota-mem-cpu.yaml -n quota-mem-cpu-example

resourcequota/mem-cpu-demo created

# 4、查看 ResourceQuota 详情:
[root@k8s-master01 ~]# kubectl get resourcequota mem-cpu-demo --namespace=quota-mem-
cpu-example --output=yaml
# 跟我们配置的一样
spec:
  hard:
    limits.cpu: "2"
    limits.memory: 2Gi
    requests.cpu: "1"
    requests.memory: 1Gi
```

ResourceQuota 在 quota-mem-cpu-example 命名空间中设置了如下要求：

- 每个容器必须有内存请求和限制，以及 CPU 请求和限制。
- 所有容器的内存请求总和不能超过1 GiB。
- 所有容器的内存限制总和不能超过2 GiB。
- 所有容器的 CPU 请求总和不能超过1 cpu。
- 所有容器的 CPU 限制总和不能超过2 cpu。

也就是在名称空间 quota-mem-cpu-example种创建Pod，必须遵守我们在上面定义的要求

5.1.1、创建 Pod

```
cat > quota-mem-cpu-pod.yaml << EFO
apiVersion: v1
kind: Pod
metadata:
  name: quota-mem-cpu-demo
spec:
  containers:
  - name: quota-mem-cpu-demo-ctr
    image: nginx
    resources:
      limits:
        memory: "800Mi"
        cpu: "800m"
      requests:
        memory: "600Mi"
        cpu: "400m"
EFO
# create Pod
kubectl apply -f quota-mem-cpu-pod.yaml --namespace=quota-mem-cpu-example

# 查看配额，能看到用了多少
```

```
[root@k8s-master01 ~]# kubectl get resourcequota mem-cpu-demo --namespace=quota-mem-cpu-example --output=yaml
spec:
  hard:
    limits.cpu: "2"
    limits.memory: 2Gi
    requests.cpu: "1"
    requests.memory: 1Gi
status:
  hard:
    limits.cpu: "2"
    limits.memory: 2Gi
    requests.cpu: "1"
    requests.memory: 1Gi
  used:
    limits.cpu: 800m
    limits.memory: 800Mi
    requests.cpu: 400m
    requests.memory: 600Mi
```

5.1.2、尝试创建第二个 Pod

```
[root@k8s-master01 ~]# cat  quota-mem-cpu-pod-2.yaml
apiVersion: v1
kind: Pod
metadata:
  name: quota-mem-cpu-demo-2
spec:
  containers:
  - name: quota-mem-cpu-demo-2-ctr
    image: redis
    resources:
      limits:
        memory: "1Gi"
        cpu: "800m"
      requests:
        memory: "700Mi"
        cpu: "400m"

# 尝试创建
[root@k8s-master01 ~]# kubectl apply -f  quota-mem-cpu-pod-2.yaml --namespace=quota-mem-cpu-example
Error from server (Forbidden): error when creating "quota-mem-cpu-pod-2.yaml": pods "quota-mem-cpu-demo-2" is forbidden: exceeded quota: mem-cpu-demo, requested: requests.memory=700Mi, used: requests.memory=600Mi, limited: requests.memory=1Gi

# 第二个 Pod 不能被创建成功。输出结果显示创建第二个 Pod 会导致内存请求总量超过内存请求配额。

# 删除你的命名空间：
```

```
kubectl delete namespace quota-mem-cpu-example
```

5.2、配置命名空间下 Pod 配额

如何配置一个命名空间下可运行的 Pod 个数配额?

```
# 1、创建一个命名空间
kubectl create namespace quota-pod-example

# 2、创建 ResourceQuota，指定改ns只可以创建2个pod
apiVersion: v1
kind: ResourceQuota
metadata:
  name: pod-demo
spec:
  hard:
    pods: "2"

# 3、apply ResourceQuota
kubectl apply -f quota-pod.yaml --namespace=quota-pod-example

# 4、查看资源配额的详细信息：
kubectl get resourcequota pod-demo --namespace=quota-pod-example --output=yaml

# 5、创建Deployment，且replicas是3，那么肯定只有2个Pod能正常运行！自己去试试吧
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pod-quota-demo
spec:
  selector:
    matchLabels:
      purpose: quota-demo
  replicas: 3
  template:
    metadata:
      labels:
        purpose: quota-demo
    spec:
      containers:
      - name: pod-quota-demo
        image: nginx
```

中文官网文档：https://kubernetes.io/zh/docs/concepts/policy/limit-range/