拉勾教育
—互联网人实战大学—

《Kubernetes 原理剖析与实战应用》

正范

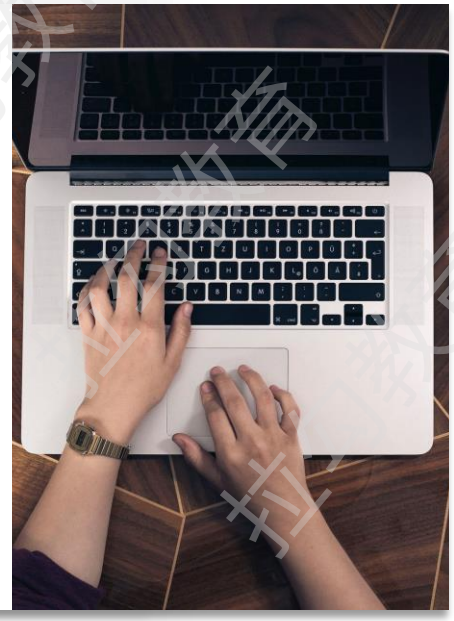
— 拉勾教育出品 —

19 | 资源限制：如何保障你的 Kubernetes 集群资源不会被打爆

前言

如果集群资源本身就是受限的情况下，或者一时无法短时间内扩容

那么我们该**如何控制集群的整体资源水位，保障集群资源不会被“打爆”？**



设置 Requests 和 Limits

CGroup 是Linux内核的一个功能

用来限制、控制与分离一个**进程组**的资源（如 CPU、内存、磁盘输入输出等）

每一种资源比如 CPU、内存等，都有对应的 CGroup

设置 Requests 和 Limits

Requests

表示容器可以得到的资源
或者可以理解为 Pod 运行的最低资源要求

Limits

表示着容器最多可以得到的资源

设置 Requests 和 Limits


```yaml
apiVersion: v1
kind: Pod
metadata:
 name: pod-resource-demo
 namespace: demo
spec:
 containers:
 - name: demo-container-1
  image: nginx:1.19
  resources:
  requests:
   memory: "64Mi"
   cpu: "250m"
  limits:
   memory: "128Mi"
   cpu: "500m"
 - name: demo-container-2
  image: nginx:1.19
  resources:
```

设置 Requests 和 Limits


```yaml
containers:
- name: demo-container-1
  image: nginx:1.19
  resources:
  requests:
    memory: "64Mi"
    cpu: "250m"
  limits:
    memory: "128Mi"
    cpu: "500m"
- name: demo-container-2
  image: nginx:1.19
  resources:
  requests:
    memory: "64Mi"
    cpu: "250m"
  limits:
    memory: "128Mi"
    cpu: "500m"
```

设置 Requests 和 Limits

```yaml
containers:
- name: demo-container-1
  image: nginx:1.19
  resources:
    requests:
      memory: "64Mi"
      cpu: "250m"
    limits:
      memory: "128Mi"
      cpu: "500m"
- name: demo-container-2
  image: nginx:1.19
  resources:
    requests:
      memory: "64Mi"
      cpu: "250m"
    limits:
      memory: "128Mi"
      cpu: "500m"
```

https://kubernetes.io/zh/docs/concepts/configuration/manage-resources-containers/#pod-%E5%92%8C-%E5%AE%B9%E5%99%A8%E7%9A%84%E8%B5%84%E6%BA%90%E8%AF%B7%E6%B1%82%E5%92%8C%E7%BA%A6%E6%9D%9F

设置 Requests 和 Limits

BestEffort

优先级最低

表示 Pod 中没有一个容器设置了 Requests 或 Limits

Burstable

优先级中等

表示 Pod 中每个容器至少定义了 CPU 或 Memory 的 Requests，或者 Requests 和 Limits 不相等

Guranteed

优先级最高

表示 Pod 中每个容器 Requests 和 Limits 都相等

简单来说就是cpu.limits = cpu.requests，memory.limits = memory.requests

一个 Burstable Pod 的例子


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: memory-burstable-demo
  namespace: demo
spec:
  containers:
  - name: memory-demo
    image: polinux/stress
    resources:
      requests:
        memory: "50Mi"
      limits:
        memory: "100Mi"
    command: ["stress"]
    args: ["--vm", "1", "--vm-bytes", "250M", "--vm-hang", "1"]
```

一个 Burstable Pod 的例子


```
$ kubectl -n demo get po
NAME                      READY   STATUS      RESTARTS   AGE
memory-burstable-demo     0/1     OOMKilled   1          11s
```

一个 Burstable Pod 的例子


```yaml
apiVersion: v1
kind: Pod
metadata:
 name: cpu-burstable-demo
 namespace: demo
spec:
 containers:
 - name: cpu-demo
  image: vish/stress
  resources:
   limits:
    cpu: "1"
   requests:
    cpu: "0.5"
  args:
  -cpus
  "2"
```

一个 Burstable Pod 的例子


```
kubectl -n demo top cpu-burstable-demo cpu-demo
NAME       CPU(cores)   MEMORY(bytes)
cpu-demo   1000m        0Mi
```

QoS 的主要作用

集群运行一段时间以后，Node 上会有很多 Running 的 Pod

Kubernetes 就会根据 QoS 的优先级来选择 Kill 掉一部分 Pod，哪些会先被 Kill 掉呢？

QoS 的主要作用

QoS	oom_score_adj
Guaranteed	-998
Burstable	min(max(2, 1000 - (1000 * memoryRequestBytes) / machineMemoryCapacityBytes), 999)
BestEffort	1000

通过 LimitRange 设置资源防线


```yaml
apiVersion: v1
kind: LimitRange
metadata:
 name: mem-limit-range
 namespace: example
spec:
 limits:
 - default: # 默认 limit
    memory: 512Mi
    cpu: 2
   defaultRequest: # 默认 request
    memory: 256Mi
    cpu: 0.5
   max: # 最大 limit
    memory: 800Mi
    cpu: 3
   min: # 最小 request
    memory: 100Mi
    cpu: 0.3
   maxLimitRequestRatio: # limit/request 的最大比率
    memory: 2
    cpu: 2
   type: Container # 支持 Container / Pod / PersistentVolumeClaim 三种类型
```

通过 LimitRange 设置资源防线


```yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: mem-limit-range
  namespace: example
spec:
  limits:
  - default: # 默认 limit
      memory: 512Mi
      cpu: 2
    defaultRequest: # 默认 request
      memory: 256Mi
      cpu: 0.5
    max: # 最大 limit
      memory: 800Mi
      cpu: 3
    min: # 最小 request
      memory: 100Mi
      cpu: 0.3
    maxLimitRequestRatio: # limit/request 的最大比率
      memory: 2
      cpu: 2
    type: Container # 支持 Container / Pod / PersistentVolumeClaim 三种类型
```

通过 LimitRange 设置资源防线


```yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: mem-limit-range
  namespace: example
spec:
  limits:
  - default: # 默认 limit
      memory: 512Mi
      cpu: 2
    defaultRequest: # 默认 request
      memory: 256Mi
      cpu: 0.5
    max: # 最大 limit
      memory: 800Mi
      cpu: 3
    min: # 最小 request
      memory: 100Mi
      cpu: 0.3
    maxLimitRequestRatio: # limit/request 的最大比率
      memory: 2
      cpu: 2
    type: Container # 支持 Container / Pod / PersistentVolumeClaim 三种类型
```

通过 LimitRange 设置资源防线


```yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: mem-limit-range
  namespace: example
spec:
  limits:
  - default:  # 默认 limit
      memory: 512Mi
      cpu: 2
    defaultRequest:  # 默认 request
      memory: 256Mi
      cpu: 0.5
    max:  # 最大 limit
      memory: 800Mi
      cpu: 3
    min:  # 最小 request
      memory: 100Mi
      cpu: 0.3
    maxLimitRequestRatio:  # limit/request 的最大比率
      memory: 2
      cpu: 2
    type: Container # 支持 Container / Pod / PersistentVolumeClaim 三种类型
```

通过 LimitRange 设置资源防线


```yaml
apiVersion: v1
kind: LimitRange
metadata:
 name: mem-limit-range
 namespace: example
spec:
 limits:
 - default: # 默认 limit
    memory: 512Mi
    cpu: 2
   defaultRequest: # 默认 request
    memory: 256Mi
    cpu: 0.5
   max: # 最大 limit
    memory: 800Mi
    cpu: 3
   min: # 最小 request
    memory: 100Mi
    cpu: 0.3
   maxLimitRequestRatio: # limit/request 的最大比率
    memory: 2
    cpu: 2
   type: Container # 支持 Container / Pod / PersistentVolumeClaim 三种类型
```

ResourceQuota 设置资源总量限制


```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources
  namespace: demo            #在demo空间下
spec:
  hard:
    requests.cpu: "10"          #cpu预配置10
    requests.memory: 100Gi      #内存预配置100Gi
    limits.cpu: "40"            #cpu最大不超过40
    limits.memory: 200Gi        #内存最大不超过200Gi
```

ResourceQuota 设置资源总量限制


```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources
  namespace: demo            #在demo空间下
spec:
  hard:
    requests.cpu: "10"           #cpu预配置10
    requests.memory: 100Gi       #内存预配置100Gi
    limits.cpu: "40"             #cpu最大不超过40
    limits.memory: 200Gi         #内存最大不超过200Gi
```

ResourceQuota 设置资源总量限制

拉勾教育
互联网人实战大学

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources
  namespace: demo            #在demo空间下
spec:
  hard:
    requests.cpu: "10"          #cpu预配置10
    requests.memory: 100Gi      #内存预配置100Gi
    limits.cpu: "40"            #cpu最大不超过40
    limits.memory: 200Gi        #内存最大不超过200Gi
```

ResourceQuota 设置资源总量限制

拉勾教育
互联网人实战大学

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources
  namespace: demo            #在demo空间下
spec:
  hard:
    requests.cpu: "10"       #cpu预配置10
    requests.memory: 100Gi   #内存预配置100Gi
    limits.cpu: "40"         #cpu最大不超过40
    limits.memory: 200Gi     #内存最大不超过200Gi
```

ResourceQuota 设置资源总量限制

拉勾教育
互联网人实战大学

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources
  namespace: demo            #在demo空间下
spec:
  hard:
    requests.cpu: "10"          #cpu预配置10
    requests.memory: 100Gi      #内存预配置100Gi
    limits.cpu: "40"            #cpu最大不超过40
    limits.memory: 200Gi        #内存最大不超过200Gi
```

ResourceQuota 设置资源总量限制


```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources
  namespace: demo            #在demo空间下
spec:
 hard:
  requests.cpu: "10"         #cpu预配置10
  requests.memory: 100Gi     #内存预配置100Gi
  limits.cpu: "40"           #cpu最大不超过40
  limits.memory: 200Gi       #内存最大不超过200Gi
```

https://kubernetes.io/zh/docs/concepts/policy/resource-quotas/#%E6%89%A9%E5%B1%95%E8%B5%84%E6%BA%90%E7%9A%84%E8%B5%84%E6%BA%90%E9%85%8D%E9%A2%9D

ResourceQuota 设置资源总量限制


```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: object-counts
  namespace: demo                    #在demo命名空间下
spec:
  hard:
    configmaps: "10"                 #最多10个configmap
    pods: "20"                       #最多20个pod
    persistentvolumeclaims: "4"      #最多10个pvc
    replicationcontrollers: "20"     #最多20个rc
    secrets: "10"                    #最多10个secrets
    services: "10"                   #最多10个service
    services.loadbalancers: "2"      #最多10个lb类型的service
    requests.nvidia.com/gpu: 4       #最多10个GPU
```

写在最后

对于一些重要的线上应用，要合理地设置 Requests 和 Limits

且最好使两者的设置相等，当节点资源不足时

Kubernetes 会优先保证这些 Pod 的正常运行

Next：《20 |资源优化：Kubernetes 中有 GC（垃圾回收）吗？》

拉勾教育

—互联网人实战大学—

关注拉勾「教育公众号」
获取更多课程信息

L / A / G / O / U