61Kubernetes 系列（五十六）理解 Kubernetes Limits and Requests

Kubernetes 系列（五十六）理解 Kubernetes Limits and Requests

欢迎关注我的公众号“ **云原生拓展** ”，原创技术文章第一时间推送。

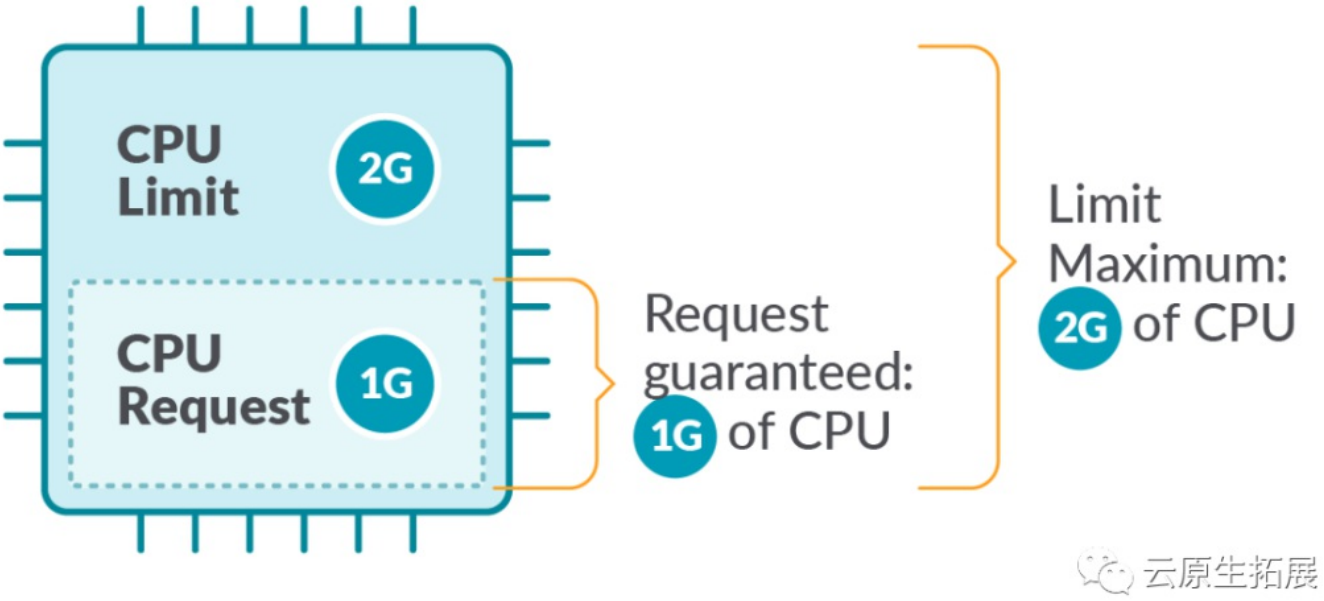
在 Kubernetes 中使用容器时，重要的是要了解涉及的资源是什么以及如何配置它们。有些进程比其他进程需要更多的 CPU 或内存。有些很关键，永远不应该挨饿。

Kubernetes Limits 和 Requests 介绍

使用 Kubernetes 时，限制和请求是重要的设置。本文将重点关注两个最重要的：CPU 和内存。

Kubernetes 将 Limits 定义为容器可以使用的最大资源量。这意味着容器永远不会消耗超过指示的内存量或 CPU 量。

另一方面，Requests 是为容器保留的最低保证资源量。

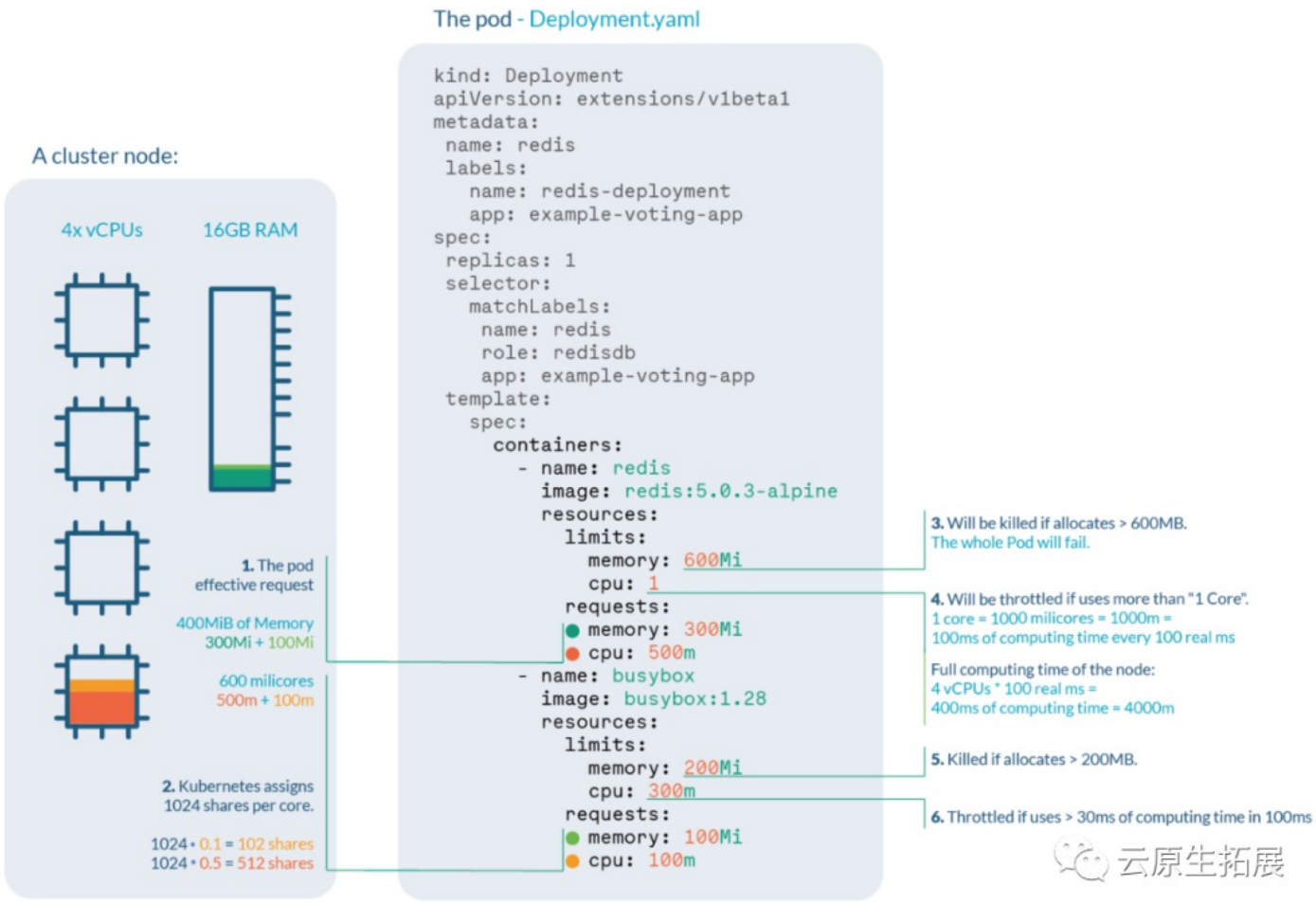


实例

让我们来看看这个部署，我们在 CPU 和内存上为两个不同的容器设置限制和请求。

```
kind: Deployment
apiVersion: extensions/v1beta1
…
template:
  spec:
    containers:
      - name: redis
        image: redis:5.0.3-alpine
        resources:
          limits:
            memory: 600Mi
            cpu: 1
          requests:
            memory: 300Mi
            cpu: 500m
      - name: busybox
        image: busybox:1.28
        resources:
          limits:
            memory: 200Mi
            cpu: 300m
          requests:
            memory: 100Mi
            cpu: 100mCode language: JavaScript (javascript)
```

假设我们正在运行一个具有 4 个内核和 16GB RAM 节点的集群。我们可以提取出很多信息：



1. Pod 有效请求是 400 MiB 内存和 600 毫核 CPU。您需要一个具有足够可用可分配空间的节点来调度 Pod。

2. redis 容器的 CPU 份额为 512，busybox 容器的 CPU 份额为 102。 Kubernetes 总是为每个核心分配 1024 个份额，因此 redis：1024 * 0.5 个核心 = 512 和 busybox：1024 * 0.1 个核心 = 102

3. 如果 Redis 容器尝试分配超过 600MB 的 RAM，则它会被 OOM 终止，很可能导致 pod 失败。

4. 如果 Redis 每 100 毫秒尝试使用超过 100 毫秒的 CPU，（因为我们有 4 个内核，可用时间为每 100 毫秒 400 毫秒），Redis 将受到 CPU 限制，从而导致性能下降。

5. 如果 Busybox 容器试图分配超过 200MB 的 RAM，它将被 OOM 终止，从而导致 pod 失败。

6. 如果 Busybox 尝试每 100 毫秒使用超过 30 毫秒的 CPU，它将遭受 CPU 限制，从而导致性能下降。

Kubernetes Requests

Kubernetes 将 requests 定义为容器使用的保证最小资源量。

基本上，它将设置容器消耗的最小资源量。

当一个 Pod 被调度时，kube-scheduler 将检查 Kubernetes requests，以便将它分配给一个特定的节点，该节点至少可以满足 Pod 中所有容器的数量。如果请求的数量高于可用资源，则 Pod 将不会被调度并保持在 Pending 状态。

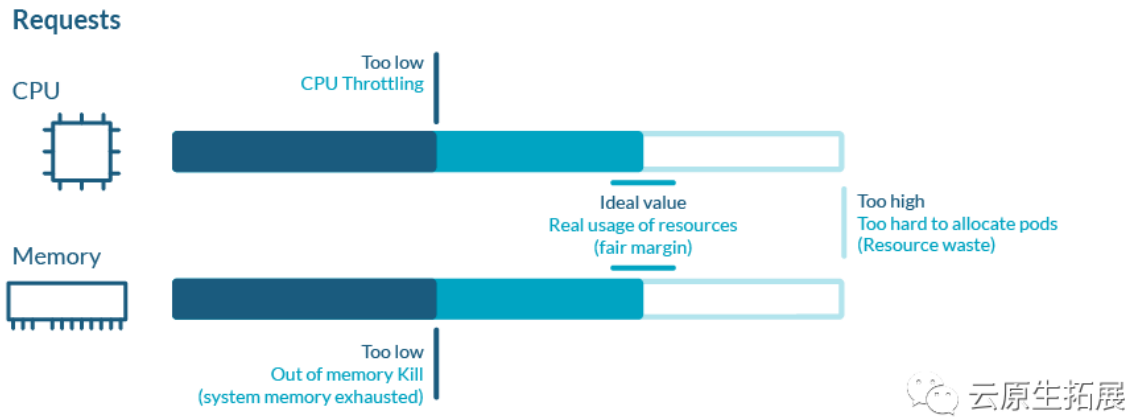
在此示例中，在容器定义中我们设置了 100m CPU 内核和 4Mi 内存的请求：

```
resources:
   requests:
        cpu: 0.1
        memory: 4Mi
```

Requests 用于:

• 将 Pod 分配给 Node 时，满足 Pod 中容器指示的请求。

• 在运行时，指示的请求量将保证为该 Pod 中的容器的最小请求量。



Kubernetes Limits

Kubernetes 将 limits 定义为容器可以使用的最大资源量。

这意味着容器永远不会消耗超过指示的内存量或 CPU 量。

```
    resources:
      limits:
        cpu: 0.5
        memory: 100Mi
```

Limits 用于:

• 将 Pod 分配给节点时。如果没有设置 requests，默认情况下，Kubernetes 将分配 requests = limits。

• 在运行时，Kubernetes 将检查 Pod 中的容器是否消耗了比 limits 中指示的更多的资源。

CPU 特性

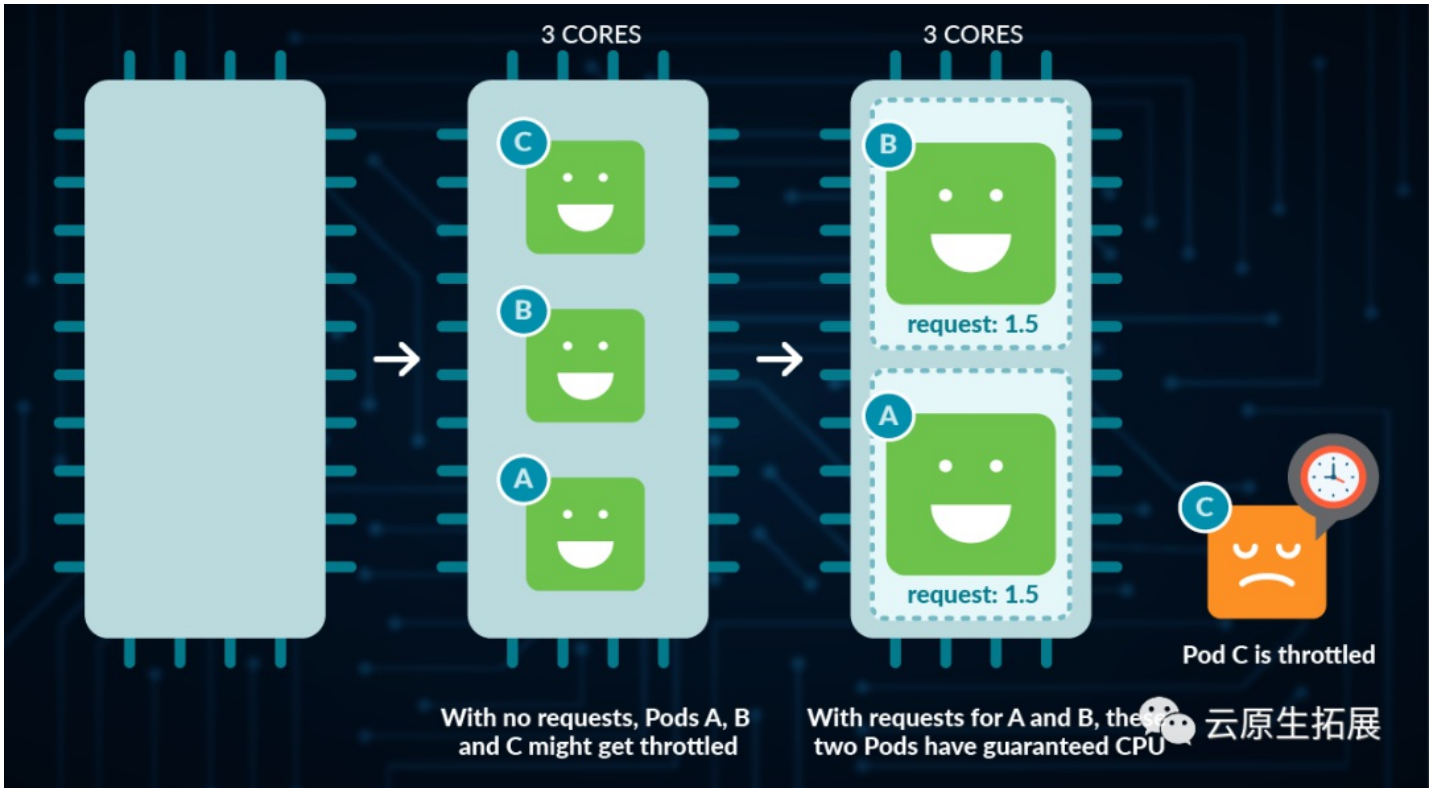
CPU 是一种可压缩资源，这意味着它可以被拉伸以满足所有需求。如果进程请求太多 CPU，其中一些将被限制。

CPU 代表计算处理时间，以内核为单位。

• 您可以使用 millicores (m) 来表示比核心更小的数量（例如，500m 将是核心的一半）

• 最小量为1m

• 一个节点可能有多个可用核心，因此请求 CPU > 1 是可能的



Memory 特性

内存是不可压缩的资源，这意味着它不能像 CPU 那样被拉伸。如果一个进程没有足够的内存来工作，这个进程就会被杀死。

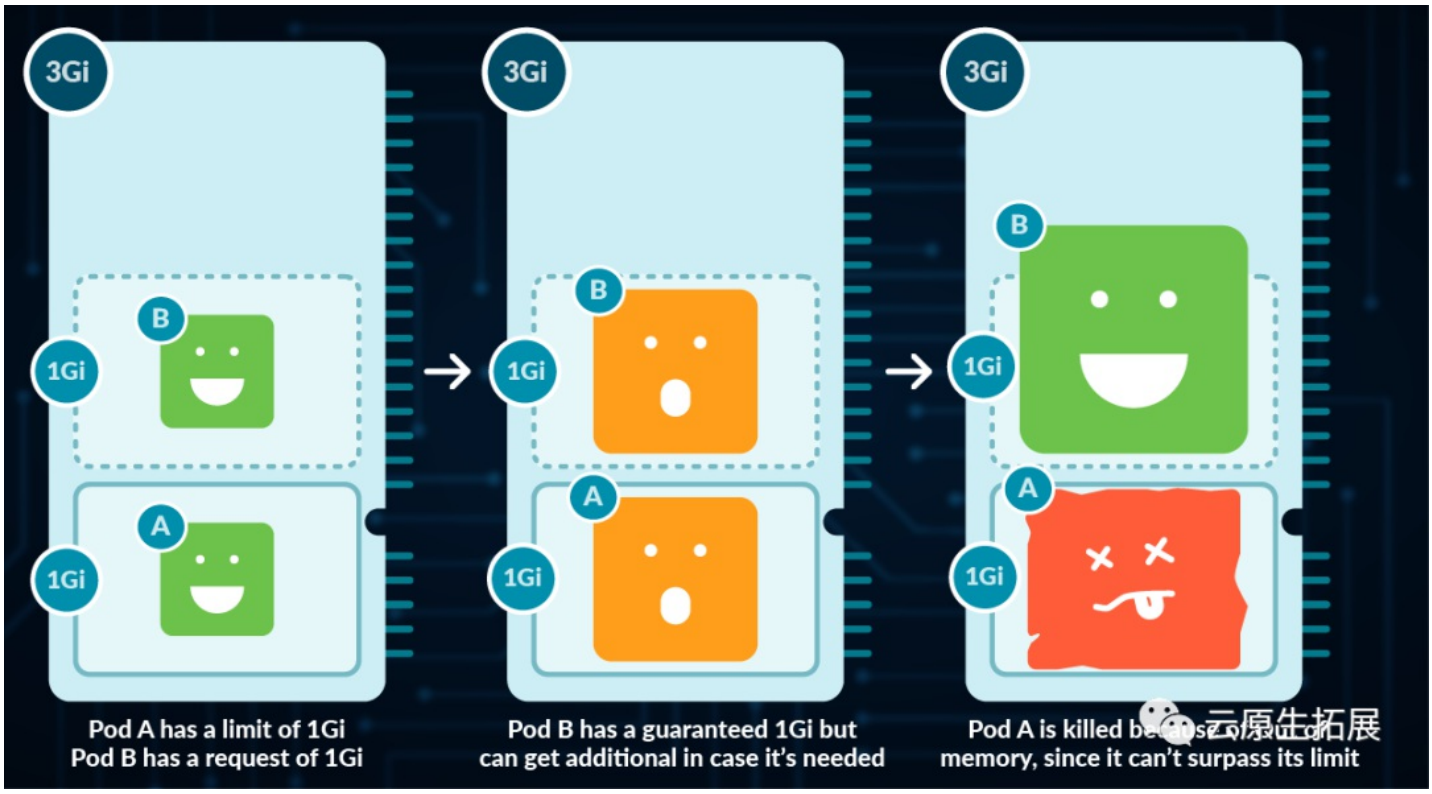
内存在 Kubernetes 中以字节为单位测量。

• 您可以使用 E、P、T、G、M、k 来表示 Exabyte、Petabyte、Terabyte、Gigabyte、Megabyte 和 kilobyte，尽管通常只使用后四种。 （例如，500M、4G）

• 警告：不要使用小写的 m 表示内存（这代表 Millibytes，低得离谱）

• 您可以使用 Mi 定义 Mebibytes，其余定义为 Ei、Pi、Ti（例如 500Mi）

1 Mebibyte（及其类似物 Kibibyte、Gibibyte 等）是 2 的 20 字节次方。它的创建是为了避免与公制的 Kilo、Mega 定义混淆。您应该使用这种表示法，因为它是字节的规范定义，而 Kilo 和 Mega 是 1000 的倍数



最佳实践

在极少数情况下，您应该使用 limits 来控制 Kubernetes 中的资源使用。这是因为如果你想避免饥饿（确保每个重要进程都得到它的份额），你应该首先使用 requests。

通过设置 limits，你只是防止进程在异常情况下获取额外的资源，在内存的情况下导致 OOM kill，在 CPU 的情况下 Throttling（进程将需要等待直到 CPU 可以再次使用）。

如果您将 requests 设置为等于 Pod 的所有容器中的限制，则该 Pod 将获得有保证的服务质量。

另请注意，资源使用率高于请求的 Pod 更有可能被驱逐，因此设置非常低的请求弊大于利。

Namespace ResourceQuota

多亏了命名空间，我们可以将 Kubernetes 资源隔离到不同的组中，也称为租户。

使用 ResourceQuotas，您可以为整个命名空间设置内存或 CPU 限制，确保其中的实体不会消耗超过该数量的资源。

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: mem-cpu-demo
spec:
  hard:
    requests.cpu: 2
    requests.memory: 1Gi
    limits.cpu: 3
    limits.memory: 2Gi
```

- requests.cpu：此命名空间中所有请求总和的最大 CPU 量
- requests.memory：此命名空间中所有请求总和的最大内存量
- limits.cpu：此命名空间中所有限制总和的最大 CPU 数量
- limits.memory：此命名空间中所有限制总和的最大内存量

然后，将其应用于您的命名空间：

```
kubectl apply -f resourcequota.yaml --namespace=mynamespace
```

您可以列出命名空间的当前 ResourceQuota：

```
kubectl get resourcequota -n mynamespace
```

请注意，如果您为命名空间中的给定资源设置 ResourceQuota，则需要相应地为该命名空间中的每个 Pod 指定限制或请求。否则，Kubernetes 将返回“failed quota”错误：

```
Error from server (Forbidden): error when creating "mypod.yaml": pods "mypod" is forbidden: failed quota: mem-cpu-demo: must speci
```

如果您尝试添加容器限制或请求超过当前 ResourceQuota 的新 Pod，Kubernetes 将返回“超出配额”错误：

```
Error from server (Forbidden): error when creating "mypod.yaml": pods "mypod" is forbidden: exceeded quota: mem-cpu-demo, requeste
```

Namespace LimitRange

如果我们想限制可分配给命名空间的资源总量，ResourceQuotas 很有用。但是如果我们想给里面的元素赋默认值会怎样呢?

LimitRanges 是一种 Kubernetes 策略，用于限制命名空间中每个实体的资源设置。

```
apiVersion: v1
kind: LimitRange
metadata:
  name: cpu-resource-constraint
spec:
  limits:
  - default:
      cpu: 500m
    defaultRequest:
      cpu: 500m
    min:
      cpu: 100m
    max:
      cpu: "1"
    type: Container
```

- `default`：如果未指定，创建的容器将具有此值。
- `min`：创建的容器不能有小于此的限制或请求。
- `max`：创建的容器不能有比这更大的限制或请求。

稍后，如果您创建一个没有设置请求或限制的新 Pod，LimitRange 会自动将这些值设置到它的所有容器：

```
    Limits:
      cpu:  500m
    Requests:
      cpu:  100m
```

现在，假设您添加了一个限制为 1200M 的新 Pod。您将收到以下错误：

```
Error from server (Forbidden): error when creating "pods/mypod.yaml": pods "mypod" is forbidden: maximum cpu usage per Container i
```

请注意，默认情况下，即使未设置 LimitRanges，Pod 中的所有容器也会有效地请求 100m CPU。

总结

为我们的 Kubernetes 集群选择最佳限制是获得最佳能耗和成本的关键。

为我们的 Pod 规模过大或投入过多资源可能会导致成本飙升。

尺寸过小或专用很少的 CPU 或内存将导致应用程序无法正常运行，甚至 Pod 被逐出。

如前所述，不应使用 Kubernetes Limits，除非在非常特殊的情况下，因为它们可能弊大于利。在内存不足的情况下，容器有可能被杀死，或者在 CPU 不足的情况下，容器可能会被限制。

对于 requests，当您需要确保进程获得有保证的资源共享时使用它们。