10Kubernetes系列（十二）资源 requests & limits

Kubernetes系列（十二）资源 requests & limits

当 Kubernetes 调度 Pod 时，需要宿主机有足够的资源来实际运行。如果在资源有限的节点上调度大型应用程序，那么该节点可能会耗尽内存或CPU资源，从而停止工作!

应用程序占用的资源也可能超过它们应该占用的资源。这可能是由于团队增加了过多的副本，而不需要人为地减少延迟(嘿，增加更多的副本比提高代码的效率更容易!)，从而导致错误的配置更改，导致程序失控，占用了100%的可用CPU。不管问题是由糟糕的开发人员、糟糕的代码还是糟糕的运气引起的，重要的是你要控制局面。

这次 Kubernetes最佳实践中，让我们看看如何使用资源 `requests` 和 `limits` 来解决这些问题。

Requests and Limits

`requests` 和 `limits` 是 Kubernetes 用来控制CPU和内存等资源的机制。requests 是保证容器可以获得的配额。如果一个容器请求一个资源，Kubernetes只会把它调度到一个能给它资源的节点上。另一方面， `limits` 确保容器不会超过某个值。容器只允许上升到极限，然后就被限制了。

一定要记住， `limit` 永远不能低于 `request` 。如果您尝试这样做，Kubernetes 将抛出一个错误，并且不让您运行容器。

`requests` 和 `limits` 是针对每个容器设置的。虽然Pods通常包含一个单独的容器，但是也经常看到Pods包含多个容器。Pod中的每个容器都有自己的 `request` 和 `limit` ，但是由于Pod总是作为一个组调度，因此需要将每个容器的 `limit` 和 `request` 加在一起，以获得Pod的聚合值。

要控制容器可以拥有的 `request` 和 `limit` ，可以在容器级别和命名空间级别设置配额。

接下来让我们看看这些是如何工作的。

Container settings

有两种资源: CPU 和 内存，Kubernetes 调度程序主要通过它们来确定在哪里运行你的 pods。可参考 `https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/` 查看详情。

典型的 Pod 资源配置可能如下所示，这个 POD 有两个容器:

```yaml
containers:
 - name: container1
   image: busybox
   resources:
     requests:
       memory: "32Mi"
       cpu: "200m"
     limits:
       memory: "64Mi"
       cpu: "250m"
 - name: container2
   image: busybox
   resources:
     requests:
       memory: "96Mi"
       cpu: "300m"
     limits:
       memory: "192Mi"
       cpu: "750m"
```

Pod 中的每个容器都可以设置自己的 requests 和 limits，最终累加。因此，在上面的例子中，Pod 的总 requests 值为500 mCPU和128 MiB内存，总 limits 值为1 CPU和256MiB内存。

CPU

CPU 资源以毫核为单位定义。如果您的容器需要两个完整的核心来运行，您可以将值设置为 `"2000m"` 。如果您的容器只需要¼个核心，那么您只需输入 `"250m"` 的值。

关于 CPU requests，需要记住的一件事是，如果您输入的值大于最大节点的核心数，则pod将永远不会被调度。假设你有一个需要四个核心的pod，但是你的Kubernetes集群由双核 vm组成，你的pod永远不会被调度！

除非你的应用程序是专门设计来利用多核（科学计算和数据库）的，否则最好的做法通常是将CPU请求保持在“1”或以下，并运行更多副本来扩展它。这使系统更具灵活性和可靠性。当涉及到CPU limits 时，事情变得有趣起来。CPU被认为是一种“可压缩”资源。如果你的应用程序开始达到你的CPU极限，Kubernetes就会开始限制你的容器。这意味着CPU将被人为限制，使你的应用程序的性能可能更差！然而，它不会被终止或驱逐。你可以使用 liveness health check 确保性能没有受到影响。

Memory

内存资源以字节为单位定义。就像 CPU 一样，如果您输入的内存请求大于节点上的内存量，pod 将永远不会被调度。与 CPU 资源不同，内存无法压缩。由于无法限制内存使用，如果容器超过其内存限制，它将被终止。如果你的 pod 由 Deployment，StetefulSet，DaemonSet，或其他类型的控制器管理，那么终止后，控制器将会再次启动新的 pod 替换。

Nodes

请务必记住，不能设置大于节点提供的资源的请求。例如，如果您有一个双核机器集群，则永远不会计划请求2.5核的Pod！

Namespace settings

在一个理想的世界里，Kubernetes 的容器设置足以处理所有事情，但世界是一个黑暗而可怕的地方。人们很容易忘记设置资源，或者一个流氓团队将 requests 和 limits 设置得非常高，并且占据了超集群的公平份额。

为了防止出现这些情况，可以在命名空间级别设置 `ResourceQuotas` 和 `LimitRanges` 。

ResourceQuotas

创建命名空间后，可以使用 ResourceQuotas 对其进行限制。ResourceQuatas 非常强大，让我们看看如何使用它们来限制 CPU 和内存资源的使用。

资源配额可能如下所示：

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: demo
spec:
  hard:
    requests.cpu: 500m
    requests.memory: 100Mib
    limits.cpu: 700m
    limits.memory: 5
```

https://storage.googleapis.com/gweb-cloudblog-publish/images/gcp-resourcequota3qo9.max-300x300.PNG

看看这个例子，您可以看到有四个部分。配置这些部分中的每一部分都是可选的。

requests.cpu 是命名空间中所有容器的最大 cpu requests 值（以毫核为单位）。在上面的示例中，您可以有50个包含10m请求的容器，5个包含100m请求的容器，甚至一个包含500m请求的容器。只要命名空间中请求的CPU总数小于500m 即可！

requests.memory 是命名空间中所有容器的最大内存 requests 值。在上面的示例中，您可以有50个包含2MiB请求的容器、5个包含20MiB 请求的容器，甚至一个包含100MiB请求的容器。只要命名空间中请求的总内存小于100MiB！

limits.cpu 是命名空间中所有容器的最大 cpu limits 值。这就像 request 一样，但是作为 Limit 使用。

limits.memory 是命名空间中所有容器的最大内存 limits 值。这就像 requests 一样，但是作为 limit 使用。

LimitRange

您还可以在命名空间中创建 LimitRange。与将命名空间视为一个整体的配额不同，LimitRange适用于单个容器。这有助于防止人们在命名空间内创建超小型或超大型容器。

LimitRange 可能如下所示：

```yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: demo
spec:
  limits:
- default:
    cpu: 600m
    memory: 100Mib
  defaultRequest:
    cpu: 100m
    memory: 50Mib
  max:
    cpu: 1000m
    memory: 200Mib
  min:
    cpu: 10m
    memory: 10Mib
  type: Co
```

看看这个例子，您可以看到有四个部分。 同样，设置这些部分中的每一个都是可选的。

default 部分配置为 POD 中的容器设置的默认 **limits** 。如果在 limitRange 中设置这些值，则任何未明确设置这些值的容器都将被分配默认值。

defaultRequest部分 为pod中的容器设置默认 **requests** 。如果在 limitRange 中设置这些值，则任何未明确设置这些值的容器都将被分配默认值。

max 部分 将设置 POD 中容器可以设置的最大 limits。default 部分不能高于此值。同样，在容器上设置的 limit 不能高于此值。重要的是要注意，如果设置了此值而未设置默认部分，则任何本身未明确设置这些值的容器都将被指定最大值作为 limit。

min 部分 设置 Pod 中容器可以设置的最小 requests。defaultRequest部分不能低于此值。同样，在容器上设置的 request 也不能低于此值。需要注意的是，如果设置了此值而未设置 defaultRequest 部分，则最小值也将成为 defaultRequest 值。

POD 生命周期

自始至终，Kubernetes 调度程序使用资源请求来运行您的工作负载，因此了解其工作原理非常重要，这样你才可以正确调整容器。假设您想在集群上运行一个Pod，并且 Pod 配置有效，Kubernetes 调度程序将轮询选择一个节点来运行您的工作负载。

注意：例外情况是，如果您使用 nodeSelector 或 类似机制来强制 Kubernetes 将 Pod 安排在特定位置。使用 nodeSelector 时仍会进行资源检查，但 Kubernetes 只会检查具有所需标签的节点。

然后，Kubernetes检查节点是否有足够的资源来满足Pod容器上的资源请求。如果没有，则移动到下一个节点。如果系统中的节点都没有剩余的资源来填充请求，则POD将进入 `"Pending"` 状态。使用 GKE 功能，如 节点自动缩放器 ，Kubernetes引擎可以自动检测此状态并自动创建更多节点。如果容量过剩，autoscaler还可以缩小并删除节点，以节省资金！但是极限呢？如你所知，limits 高于 requests，如果您有一个节点，其中所有容器 limit 的总和实际上高于机器上可用的资源，该怎么办？

在这一点上，Kubernetes 进入了一种被称为"过度承诺"的状态。这就是事情变得有趣的地方。由于CPU可以压缩，Kubernetes 将确保容器获得其请求的CPU，并限制其余容器的CPU。内存无法压缩，因此Kubernete 需要开始决定在节点内存不足时终止哪些容器。

让我们想象一个场景，在这个场景中，我们有一台内存不足的机器。Kubernetes 会怎么做？

注：以下适用于Kubernetes 1.9及以上版本。在以前的版本中，它使用了稍微不同的过程。

Kubernetes 会寻找那些实际占用比他们配置的 requests 值更高的 Pods，如果你的 Pod 容器没有请求量，那么默认情况下，如果它们占用的比请求的多，那么这些POD是将被终止的主要候选对象。其次主要候选容器是已经超过其请求但仍在其限制范围内的容器。

如果 Kubernetes 发现多个 Pod 已经远远超过了他们的请求量，那么它将根据Pod的[优先级]对其进行排序，并首先终止最低优先级的Pod。如果所有Pod具有相同的优先级，Kubernetes会终止超出其请求最多的Pod。在极少数情况下，Kubernetes可能会被迫终止仍在其请求范围内的pod。当关键系统组件（如kubelet或docker）占用的资源超过为其保留的资源时，就会发生这种情况。

总结

虽然您的 Kubernetes 集群可能在不设置资源请求和限制的情况下运行良好，但随着团队和项目的增长，您将开始遇到稳定性问题。向 pod 和 命名空间 添加 `requests` 和 `limits` 只需要一点点额外的努力，并且可以避免您在后续工作中遇到许多麻烦!