|  |  |
| --- | --- |
| 交底书名称 | 一种基于kubernetes的容器驱逐策略 |
| 技术联系人姓名 | 马殿军 |
| 技术联系人电话 | 13269325833 |
| 技术联系人Email | madianjun@jd.com |

（技术联系人信息用于与外部代理沟通，发明人信息在ERP专利申请系统中填写）

注意事项：

1、代理人并不是技术专家，交底书要使代理人能看懂，尤其是完整技术方案，一定要写得全面、清楚。

2、在后续与专利代理人进行沟通时，对于代理人的疑问应认真讲解，要求补充的材料应及时补充（禁止通过私人邮箱与代理人沟通）。

3、常用检索网站：www.soopat.com（SOOPAT），http://so.baiten.cn/（佰腾），patents.google.com（谷歌专利）。

# 1. 现有技术

/\* 应记载某个应用场景或者解决某个技术问题当前所采用的技术，可以概述该技术，也可以仅给出参考文献的链接或相关专利号。

Kubernetes是一种容器集群编排和管理的分布式系统，它将容器调度并运行在集群的多个节点上。当容器由于某些原因（物理机问题或容器内部程序问题）导致无法正常运行时，kubernetes会反复重启容器，直到容器可以正常运行为止。

# 2. 现有技术的缺点

/\* 需要指出现有技术存在的缺点，本发明也不能克服的缺点无需提供。

/\* 应根据现有技术的实现过程，有针对性地说明缺点产生的原因。

当容器无法成功运行时，kubernetes会在同一个节点上反复重启容器。如果由于物理机自身的问题导致容器无法运行，那么在同一个节点上反复重启容器是没有作用的，因为容器运行所依赖的外部环境并没有改变。物理机自身的问题包括网络故障、硬盘故障等。这时就需要一种策略让容器被调度到其他节点上，避开当前节点的环境问题。

# 3. 本发明技术方案

3.1 本发明所要解决的技术问题（即发明目的）

/\* 描述本发明所要解决的技术问题，与“2.现有技术的缺点”部分指出的缺点相对应。

本发明主要解决的是让无法正常运行的容器被重新调度，并以一定概率被调度到其他节点上，避免容器在同一个节点上反复重启的问题。

3.2 本发明的完整技术方案的详细阐述

/\* 这是本文档最重要的部分，需要详细完整的阐述，不能光有原理，也不能仅有功能性介绍或操作说明。

/\* 在描述具体的技术方案时，必须结合附图（方法型专利按照数据流向或实现步骤抽象框图，装置型专利按照组成部件抽象框图）进行说明，每个附图都应当有对应的文字描述。如果本发明方案包含多个主题，方法与装置等，则需要分别进行描述。

1. 架构

本发明在基于kubernetes原有系统架构的基础上，增加一个descheduler组件。descheduler定期从apiserver上获取所有pod的信息，并过滤出已经被调度到node上但没有正常运行的pod，并将其驱逐，进而触发scheduler的对新pod的重新调度，新产生的pod将以一定概率被调度其他node上，因此可从一定概率上实现使pod不在同一个节点上反复重启。架构图如下：

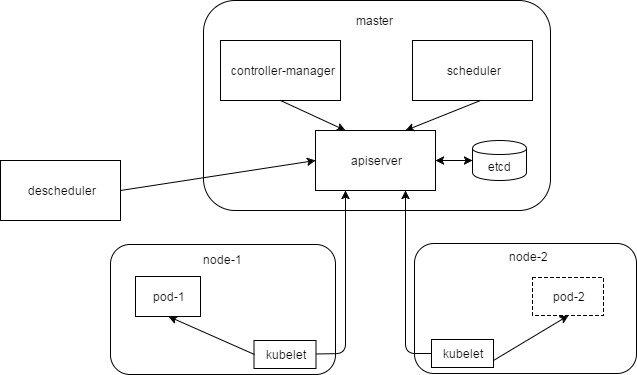


图1 descheduler与kubernetes交互架构图

详细流程描述如下：

1. pod-1在node-1上启动，但并没有成功运行，kubelet将其状态上报至apiserver。
2. descheduler定期从apiserver上获取pod列表，并检查pod状态。当发现pod-1处于“已经被调度但未成功运行”的状态时，向apiserver发出evict驱逐请求。
3. node-1上的kubelet接收到evict请求，将pod-1从node-1上删除，并将状态同步到apiserver。
4. controller-manager监听apiserver时，发现pod-1被删除，就创建一个pod-1的替代副本，称为pod-2，并将pod-2同步到apiserver。
5. scheduler监听apiserver时，发现pod-2虽然被创建了，但是没有被调度，则执行调度算法选择一个合适的node（该node可能是node-1或node-2）与pod-2绑定。
6. 如果pod-2与node-2绑定，则node-2上的kubelet从apiserver上监听到此状态变更，就在node-2上创建pod-2。

以上过程即实现了将pod-1从node-1驱逐并重新调度的过程。需要注意的是：

1) descheduler无法区分pod-1未成功运行的原因是由于node-1的物理机环境问题导致，还是由于pod-1自身的程序问题导致。descheduler是在不严格区分这两种原因的情况下触发的pod的驱逐。

2) 如果第(5)步中scheduler选择了将node-1与pod-2绑定，则pod-2会在node-1上创建，那么pod-2仍然无法正常运行。假设scheduler执行重新调度时，每个节点被选择的概率相同，那么重调度策略在一定概率上保证pod-2被调度到其他节点;集群中节点数量越多，pod-2被调度到其他节点的概率越大。

1. 检测pod未成功运行的方法

descheduler在检查pod是否应当被驱逐时，需要判断pod的状态是否满足未成功运行的条件。pod的状态字段包括以下三个（值为true或false）：

1. scheduled：如果pod被调度成功，则置为true；否则置为false。
2. initialized：pod中的容器分initial和regular两种，initial容器先运行，当initial容器运行成功并结束后，regular容器再启动。如果initial容器成功结束，则initialized被置为true。
3. ready：如果regular容器运行成功，则被置为true；如果运行失败，则被置为false。

pod的运行流程包括以下步骤：

1. 创建pod
2. 调度pod
3. 下载initial容器的镜像
4. 运行initial容器
5. 下载regular容器的镜像
6. 运行regular容器

在图2中，描述了以上步骤的执行流程，以及状态字段的变化。基于这些状态变化，pod的执行阶段在整体上被划分为两个阶段：Pending、Running，这两个阶段在图2中被标记了出来。

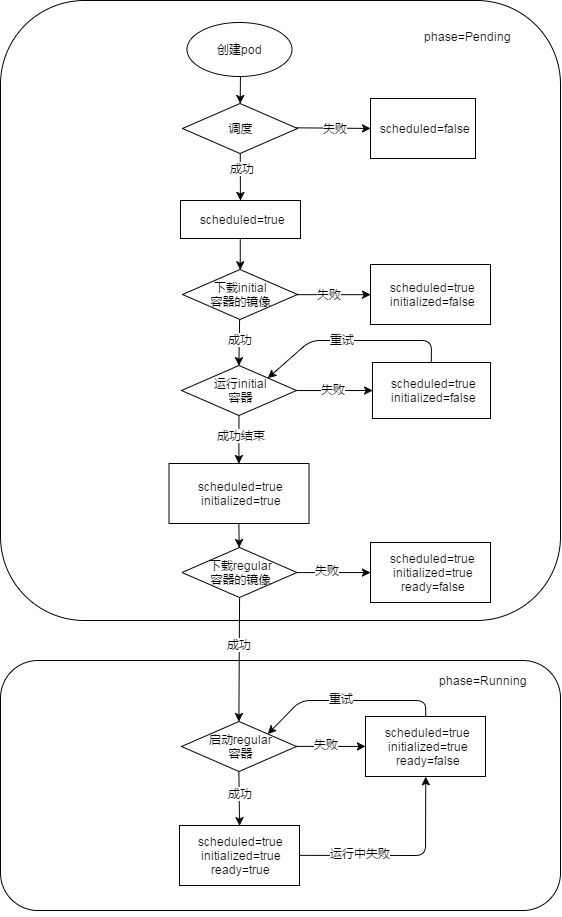


图2 pod运行流程及状态变化

descheduler所要检查的pod运行失败情况包括以下4种,同时给出了对应的状态字段的值：

1. 下载initial容器的镜像失败(scheduled=true,initialized=false)
2. 运行initial容器失败(scheduled=true,initialized=false)
3. 下载regular容器的镜像失败(scheduled=true,initialized=true,ready=false)
4. 运行regular容器失败(scheduled=true,initialized=true,ready=false)

descheduler只需要判断以上4种情况的状态字段的取值，即可检测出未成功运行的pod。

3.3 本发明希望保护的技术创新点

/\* 指出技术方案中希望保护的技术关键点，并概括说明该关键点的技术原理。

本发明的技术创新点是根据pod状态字段的取值筛选出未成功运行的pod，进而向kubernetes发送evict请求，驱逐pod并使其被重新调度。

3.4 针对3.3中的技术方案，是否还有别的替代方案同样能完成发明目的？

/\* 替代方案可以是完整技术方案的替代，也可以是部分结构或者步骤的替代。

3.5交底书中技术术语的名词解释

/\* 记载交底书中出现的专业技术术语、缩写、外文的解释。

node：集群中的节点，对应一台物理机。

kubernetes：由google开源的容器集群编排和管理系统。

master：kubernetes的控制节点，包含apiserver, controller-manager, scheduler三个组件。

etcd：一种分布kv数据库，为kubernetes提供数据持久化服务。

apiserver：为kubernetes其他组件提供api访问，它直接读写etcd。

controller-manager：维护pod副本数量，使之与用户期望的数量一致。

scheduler：调度pod，使pod与合适的node绑定。

kubelet：管理和监控本节点上的pod。

pod：是容器的一种抽象表示，它可以包含一个或多个容器实例。Kubernetes以pod作为最小的管理和调度单位。

evict：驱逐，即将pod从node上删除。