# **如何提高K8s的可恢复性**

文: Tim Little Aug 6， 译：付文新

本文原始链接地址：

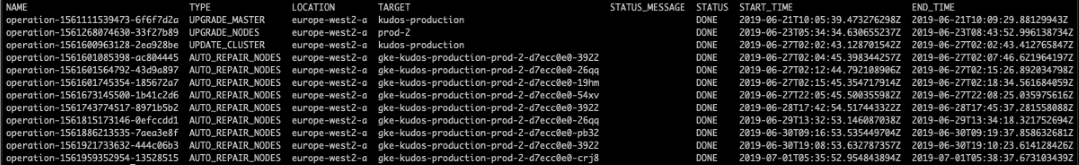
<https://medium.com/kudos-engineering/increasing-resilience-in-Kubernetes-b6ddc9fecf80>

K8s有两大关键特性：高可用性和可恢复性。但是当K8s系统不在稳定的时候，我们能做什么呢？眼睁睁看着自己的巨轮沉入深海？

K8s的节点问题

我们在使用谷歌K8s引擎集群（GKE）时常会遇见这个问题：集群中的K8s节点经常性的宕机。其影响是调度问题节点的容器变得无法响应，我们的监控系统开始巨量发送SLO和错误预算消费的告警邮件。想更多的了解什么是SLO和错误预算告警，以及我们是如何监控的，请找一下我以前的博客。

通过运行 gcloud 容器操作列表命令，我们发现节点会每隔几小时就尝试自动修复。



然而当我们查看Stackdriver日志，我们发现了海量的日志，这使得问题定位变得如同海底捞针。因此我们申请了google的支持，以期帮助我们找到问题的根本原因。可惜的是由于时差关系和我们的账号使用的是遗留支持包的原因，我们每隔24小时才能看到一条google反馈的信息。

最终这个问题耗尽了我们K8s中一系列服务的错误预算，因此在真正的SRE中，我们致力于提高K8s集群和Istio网格的可恢复性。

增加K8s服务的可恢复性

我们从深入研究自己的K8s集群设置和运行在它之上的服务开始，然后圈定我们可以增加可恢复性的范围，主要目标是减少当单节点无法响应时产生的影响。

副本数量和pods在不同节点的分发

能够提升的第一个方面的是我们K8s部署的副本数量。我们在Kudos上部署采用了微服务架构的服务，这些服务都特别微小，没有状态，特别适合水平拓展。然而有些服务只部署在一个pod上，如果这个pod刚巧部署在发生问题的节点上，我们的服务就会中断。因此我们更改了部署，运行至少3个副本以提高可恢复性。当时我们只有3个节点（现在14个），这导致了另外一个问题：有时K8s会调度同一个节点上的所有pod。因此我们看上了K8s的一个名为Pod AntiAffinity的特性：如果某个节点已经包含具有相同标签的pod，该特性将指示调度程序把此pod标记为不可用。

示例如下：

app=<service-name>.

affinity:

podAntiAffinity:

preferredDuringSchedulingIgnoredDuringExecution:

- weight: 100 podAffinityTerm: labelSelector: matchExpressions: - key: app operator: In values: - <service-name> topologyKey: "kubernetes.io/hostname"

添加了这些内容并重新部署了所有服务之后，我通过运行kubectl get pods -o customcolumns=’NAME:.metadata.name,NODENAME:.spec.nodeName’ 命令验证了pods被调度到不同的节点上。

现在如果我们的一个节点再次宕机，我们也只损失了部署的3个pod中的一个，我们的服务仍然在运行。

就绪探针与预停命令

我们注意到的另一个问题是当修复节点的时候，K8s会在节点准备就绪之前就开始发送信息。在升级K8s节点池的时候也遇到过类似问题，但我们从未深究其因。

经过google一番之后，我们发现一篇关于零宕机滚动更新的博客。这篇博客的主要内容之一是关于把pod标记为终止，并从负载均衡中将其删除的异步特性。这种重新配置是异步进行的，因此不能保证其按照正确的顺序，从而导致不少请求被分发到终止的pod上，进而致使请求崩溃。

从此以后，我们在所有的服务上都增加了就绪探针，并增加预停命令：该命令会等待10秒后终止，以允许这些少数连接仍然可用。

示例如下：

readinessProbe:

httpGet:

path: /readiness

port: 8080

initialDelaySeconds: 15

lifecycle:

preStop:

exec:

command: ["/bin/sh","-c","sleep 10"]

Istio水平自动伸缩

随着K8s的进一步稳定，我们注意到集群中另一个单点失败的工具：Istio。我们把Istio当做谷歌K8s引擎（GKE）的附加组件使用，这使得我们可以使用Istio，并且不需要额外的操作开销来管理Istio的控制面板。

Istio使我们能够按路线分发流量到网络中的任意服务中去，而不需要每个服务都有一个公共的IP地址。它通过入口网关（Ingressgateway）来实现这一点，该网关是一个位于网络边缘的特使（Envoy）网关，并且连接到google的负载均衡器上，以便获取访问外部网络的权限。

我们遇到的问题是谷歌K8s引擎（GKE）附加组件默认的Istio只在一个pod中部署入口网关（Ingressgateway）。这就意味着一旦该节点忽然宕机，我们就无法访问网络中的所有服务。这肯定不是理想方案，因此我们研究了下Istio的设置，结果发现google早就意识到这一点，可以通过部署HorizontalPodAutoscaler来修改Istio的控制面板。为此，我们需要用以下命令向Istio控制面板组件添加资源请求：

kubectl edit -n istio-system Deployments/istio-telemetry

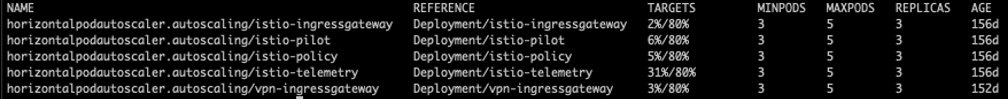
之后在容器中添加以下内容：

resources:

requests:

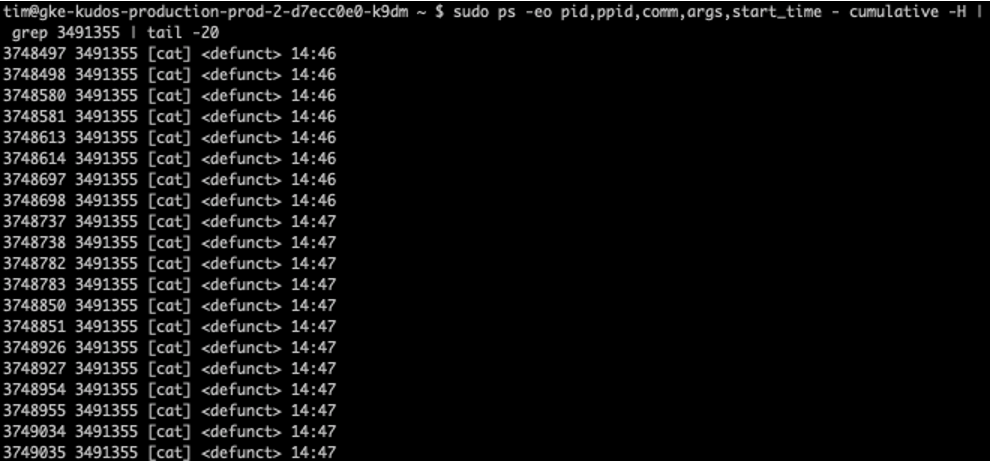
cpu: 100m

这使得HorizontalPodAutoscaler可以检测Pod的CPU利用率，并相应的进行伸缩或扩展。就如同我们之前为其他服务所做的一样，我们还编辑了HorizontalPodAutoscaler，使其至少有3个副本在运行。



根源分析

在处理可恢复性问题的时候，我们一直和google的支持团队保持联系。支持团队建议我们深入查看下K8s的节点，并怀疑这是一个资源耗尽的问题。在对节点进行了一些故障排除之后，我们发现一些节点的进程中出现了泄露，进程树中数以千计的<defunct>进程没有被正确终止。



检查了父进程的ID之后，我们发现了自己的问题所在。





3491355进程是我们其中的一个容器，它用来转换HTML页面到PDF，使用的是google的Chrome实例来实现这一点。这一服务有一个名为readiness用来检查google Chrome版本的程序，但是该程序只抓取了版本信息文件，进程并没有被正确的结束。我们之后修改了程序readiness，并重新部署了pdfgenerator服务，然后我们看到运行该服务的节点中PID数量大幅减少。

K8s开发团队预见了这个问题，并在1.14引入了PID限制。

可惜的是我们生产环境运行的1.13版本，没能够赶上这个变化。

结论

在这些问题背后，我们学到了大量关于K8s和Istio的知识。我们更新了自己的微服务模板，使部署到我们集群中的默认微服务更有可恢复性，而且扩展了集群以帮助支持新服务。

如果你对这些都感兴趣，为什么不考虑加入Kudos？？？