### 问题现象

TCE 平台某用户发现容器的 Health Check 脚本执行缓慢，耗时 0.5s ～ 2s，脚本进程的 CPU 使用率瞬时值接近 100%，而同样的脚本在宿主机执行仅需 ～0.02s。协助用户使用雨滴平台的通用性能分析功能，快速定位出问题原因并提出了可行的性能优化方案，解决了这一困扰用户近一周的疑难问题。

我们对该实际案例做了简化和模拟，在公有云 TKE 平台上，通过以下步骤可以复现、分析与优化其中的关键性能瓶颈，并展示了雨滴平台的典型使用方式。

### step1.  问题复现

执行以下步骤，启动优化前的应用 demo。

1# 启动优化前应用 demo

2kubectl create -f https://drop-user-guide-1302374296.cos.ap-shanghai.myqcloud.com/configmap-envs.yaml

3kubectl create -f https://drop-user-guide-1302374296.cos.ap-shanghai.myqcloud.com/slow-health-check.yaml

启动后，未施加负载的情况下，在 TKE 监控上观察到 Pod 的 CPU 使用量接近 0.3 核。

登入容器，执行 top，发现进程 [health\_check.sh](https://health_check.sh/)  CPU 使用率瞬时值接近 100%。

在容器中手工执行 [health\_check.sh](https://health_check.sh/)，如下图所示，耗时 1.453s，复现了 TCE 平台用户遇到的问题。

但是 [health\_check.sh](https://health_check.sh/) 脚本非常简单，内容如下所示，只有一行 echo。

1#!/bin/bash

2

3echo "health check done."

为什么执行这个脚本这么慢呢？接下来我们开始使用雨滴进行性能分析。

### step2. 雨滴性能分析

在容器中继续执行以下步骤，启动雨滴 agent（**临时方案，待改为 sidecar 更便捷的方式**）。执行结束后，【查看】任务结果。

#### 分析建议

【任务结果】->【分析建议】标签页，有一条与 Health Check 直接相关的分析建议：**进程 health\_check.sh 占比整体 74.51%。进程内 maybe\_make\_export\_env 函数占比 98.16%，可能是环境变量过多导致，请进一步确认。**

点击【火焰图】链接，查看进程 [health\_check.sh](https://health_check.sh/) 热点函数 maybe\_make\_export\_env 对应的火焰图，如下图所示，即引发进程 [health\_check.sh](https://health_check.sh/) CPU 使用率较高的代码路径。

#### 关键线索

接下来，我们根据上述【**可能是环境变量过多导致，请进一步确认】**这一提示继续分析。

在容器中执行 env 命令，统计环境变量具体数量，如下图所示，共计 10032 个环境变量，其中 1w 个都是以 KUBERNETES\_SERVICE\_ 为前缀的环境变量，这里模拟了 TCE 平台大量 service 环境变量注入情况，他们当时有接近 1w 个环境变量。

#### 快速验证

既然确认环境变量过多（1w 个），我们想到了一个快速验证这是否为问题主因的方法：在容器中执行 env -i ./health\_check.sh，清除环境变量的前提下，耗时仅 0.009s。而直接执行 ./health\_check.sh，耗时 1.526s。对比结果如下图所示，**说明 Health Check 脚本执行缓慢，正是由于环境变量过多引起。**

### step3. 性能优化方案

针对环境变量过多（1w）引发的问题，接下来该如何优化呢？这里给出 2 个具体方案，其中方案 1 由性能专家进一步分析火焰图 & bash 源码后提供（部分工作后续将借助雨滴 AI 性能知识库解决），方案 2 是 TCE 用户受到分析建议【环境变量过多】启发后想到。

**雨滴平台提供了性能分析建议，AI 进一步增强性能分析与优化能力。最后，用户还可以将雨滴任务结果链接分享出来，由性能专家开展深度分析，提供类似方案 1 的优化建议。**

#### 方案 1

查看雨滴【任务结果】【火焰图】中的热点函数 maybe\_make\_export\_env，沿着调用栈自底向上看：main -> ... -> reset\_local\_vars -> maybe\_make\_export\_env -> ...

其中，reset\_local\_vars 与 locale 相关，进一步触发了环境变量相关的函数 maybe\_make\_export\_env，那么是否跟 locale 设置有关呢？

容器中执行 locale 命令，输出如下图，其中 LC\_ALL 变量为空。经查阅 bash 源码，确认当 LC\_ALL 变量为空时，将会触发代码路径：reset\_local\_vars -> maybe\_make\_export\_env -> ...

执行 export LC\_ALL=en\_US.UTF-8，手工导出 LC\_ALL 变量，然后执行 [health\_check.sh](https://health_check.sh/) 脚本，耗时降低至 0.02ms 以下。

因此，可以在构建镜像时，通过显示设置 locale 相关的 LC\_ALL 变量，即使环境变量过多，也可以有效避免该问题的发生。

#### 方案 2

TCE 平台用户经进一步分析，发现接近 1w 个环境变量绝大多数由 K8s 注入，且大多数没有必要注入到容器中，将环境变量精简之后，Health Check 脚本的执行耗时从 0.5s ～ 2s 降低到 ~0.02ms，CPU 使用率从 ～50% 下降到 < 2%。

### step4. 优化前后对比

接下来，我们模拟 TCE 平台用户的优化方案 2，将容器环境变量从 1w 个精简至 1000 个。

#### 优化实施

执行以下步骤，首先删除优化前的应用 demo，然后再把优化后的启动。

1# 删除优化前应用 demo

2kubectl delete -f configmap-envs.yaml

3kubectl delete -f slow-health-check.yaml

4

5# 启动优化后应用 demo

6kubectl create -f https://drop-user-guide-1302374296.cos.ap-shanghai.myqcloud.com/configmap-envs-2.yaml

7kubectl create -f  https://drop-user-guide-1302374296.cos.ap-shanghai.myqcloud.com/slow-health-check-2.yaml

然后再按照上述 step2 操作，获取优化后的雨滴平台分析结果。

#### 分析建议对比

优化后，环境变量过多提示、占比超 5% 的热点函数\_\_strcmp\_avx2、vlist\_add 均被消除。

#### 执行耗时对比

优化前 0.5s ～ 2s，优化后 ～0.02s，执行缓慢现象消除。

#### CPU 使用量对比

优化后，CPU 使用量从 ～0.3 核 下降至 ～0.005 核。

至此，该性能问题得以解决。