用户故事 | 莫名：相信坚持的力量，终会厚积薄发

2021-02-10 莫名同学

容器实战高手课 进入课程 >

讲述：正霖

时长 11:17 大小 10.34M

你好，我是莫名同学，坐标杭州。

我已经工作六年多了，曾经参与过对象存储、大数据等产品的开发。最近两年，我开始接触容器云平台系统开发与性能优化方向的工作。

最初是偶然的一次机会，我关注了 eBay 技术荟公众号，认真拜读过作者的 eBay 云计算网事系列文章，受益匪浅，从中汲取了不少排查网络疑难杂症的新思路。但意犹未尽，一直期待有机会看到作者更多的分享。

后来，又看到极客时间**《容器实战高手课》**的课程预告，作者正是程远老师，而且课程内容也和我现在做的工作不谋而合。我满怀期待地翻了一遍开篇词，激起了不少共鸣，于是毫不犹豫地购买了该专栏。

我比较赞同作者“一个态度，两个步骤”的方法论。可以说，近年来我也是这个方法论的忠实践行者。正如作者所言，**容器的大多数问题，最终都可以映射到 Linux 操作系统相应模块上。**因此，想要在容器方面技高一筹，意味着需要苦练内功，深入了解隐藏在容器背后的 Linux 操作系统知识原理。

记得我刚参加工作的前几年，我没有直接接触过容器技术，对 Linux 了解也不多，仅仅停留在使用层面，遇到问题经常一头雾水。

庆幸的是，从 2017 年末起，我开始沉下心来学习 Linux 相关技术，从基础知识到内核源码，一步一步地向前进。这个过程中，也逐渐养成了遇到问题刨根问底的思维习惯，而这种态度，也正是这个专栏里一直倡导的。

2019 年初，我开始真正接触容器技术，凭借平时积累了比较扎实的 Linux 知识功底，遇到的多数容器问题都能够迎刃而解，“两个步骤”在平时的系统性能分析与优化工作中得以体现。

我为什么要学习专栏?

选择学习专栏的主要原因有两个，一方面是完善知识体系，另一方面是拓宽自己的技术思路。

经过前期了解，我认为作者技术经验丰富。他经历了 Kubernetes 容器云平台在 eBay 的具体落地过程，解决过大大小小的各类问题，是货真价实的容器实战高手。在学习专栏之前，我自己简单整理过一套容器知识结构体系。不过相比之下，作者列出的容器知识结构体系更加完备。

所以，我期待借鉴作者的思路重新审视自己的理解，查漏补缺、温故知新，并在专栏更新过程中通过留言和作者进一步交流心得体会。相信经过系统的学习，我也能更好地建立自己的知识架构。

另外，既然这门课是实战课，自然少不了动手实战，我期待可以在案例中，借鉴老师解决问题的思路。

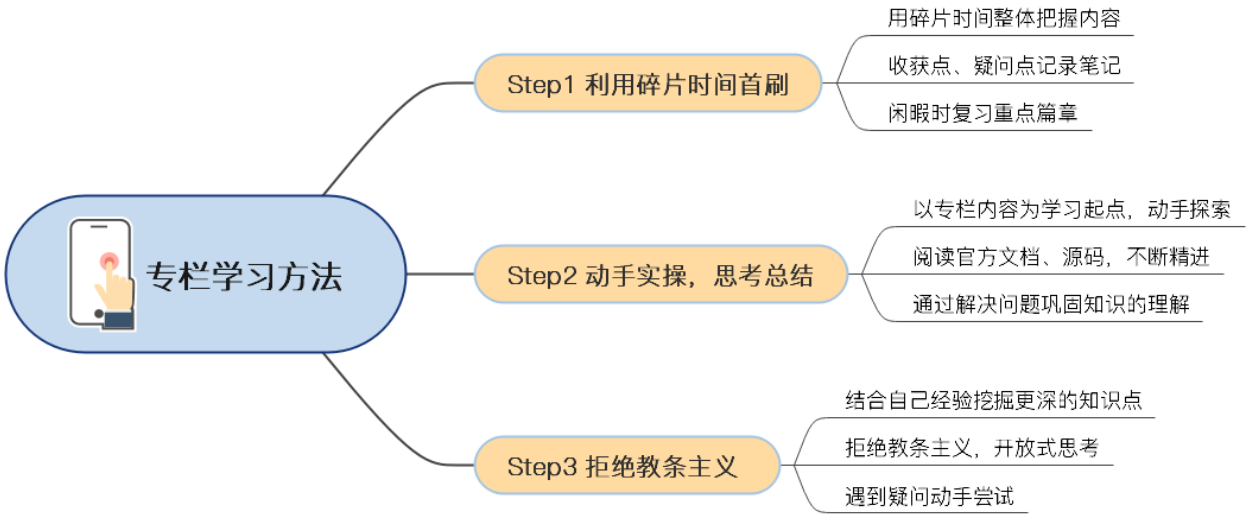
作者在 eBay 云计算网事系列文章中曾提过，Kubernetes 容器云平台在大规模应用场景下的偶发性问题十分棘手，如果手忙脚乱地进行各种尝试，可能会事倍功半、越理越乱。事

实的确如此，在复杂的 Linux 环境下，传统的性能分析工具应对这些偶发性问题，已经略显捉襟见肘了。

过去一年里，我也曾使用 perf、ftrace、BCC/bpftrace 等 Linux 性能分析工具解决过网络抖动、存储写入延迟等偶发性问题。但是仍然担心自己的思路存在局限性，所以很期待作者专题加餐的分享，看看这些工具在容器网络不稳定的真实案例中要怎么综合运用，拓宽自己定位、解决类似问题的思路。

我是怎么学习专栏的

我应该是订阅比较早的同学了，现在想想，能够坚持学完整个专栏，而且在更新期间跟老师做了积极互动，主要有这样三个方法。



1. 坚持学习，用好碎片时间

专栏的首刷，我基本上是利用碎片化的时间完成的。每周一、三、五专栏更新时，我会在早上通勤路上先把内容过一遍，从整体上把握文章思路。

由于碎片时间很难保持较高专注度，我倾向于选择轻松一些的阅读方式，不刻意做过多的深度思考，但是会把自己比较赞同或疑惑的知识点先标记下来，以便后续重点学习。

专栏内容结构简练、清晰，一共二十多篇，其实我觉得坚持下来并不难。专栏更新过程中，我利用工作闲暇时，针对最有收获的几篇文章反复刷了若干次，结合文章示例动手实践，并翻阅相关的 Linux、Docker 源码，将首刷时留下的疑点逐一击破。

两个月下来，这样的学习让我受益匪浅，进一步拓宽了自己的容器知识结构体系，也学到了分析常见容器性能问题的新思路。

我相信，当一个人开始把坚持学习当作一种习惯，并且善于合理利用碎片时间提升自我的时候，他已经走在超越大多数人的道路上。

2. 知行合一，善于思考总结

每个专栏作者几乎都会反复强调：学习专栏时，请动手实践文章中提到的示例以及课后思考题。所以，学习中思考与实践的重要性不言而喻。

数学家华罗庚说过，他的学习方法很简单，就是“读书要从薄到厚，再从厚到薄”。其实专栏的学习也是同样的道理。在我看来，专栏每篇文章都浓缩着作者历经磨砺后的思考与总结，有些很难一次性消化，所以需要我们在不断实践与思考中领悟。

我觉得，专栏起到的是一个**领航**的作用，想要相关的知识点我们可以根据自己的需要继续做功课，这样才能不断精进。遇到有疑问的地方，我会通过官方文档、源码等方式深挖这些知识点背后原理，希望可以由点及面，直达问题本质。

有了这样的问题导向，我才会进行更多深度思考，在研究、解决问题的过程中，也加深了自己对知识的理解。下面举几个我曾经思考并想办法弄清楚的问题：

什么是容器绑定挂载？容器创建过程中涉及哪些挂载行为和传播方式？

为什么说虚拟网络设备 veth 的行为像一根网线？

如何使用 strace 追踪容器创建的大致过程？

拥有独立 Network Namespace 的容器如何反过来操控宿主机的防火墙、路由表等？

为什么 Pid Namespace 可以使得容器的 init 进程的 PID 总是为 1？

…

3. 敢于怀疑，拒绝教条主义

延续刚才说的知行合一。其实我们学习的时候，对有疑问的内容拒绝教条主义，这也很重要，我举个例子吧。

第 12 讲学习容器 Quota 技术的时候，文章里用到的例子是 xfs quota。但我认为，目前容器 quota 技术存在以下局限性：容器通常采用 overlay 存储驱动，仅支持在宿主文件系统 xfs 上开启 quota 功能，这意味着 overlay over ext4 不支持 project quota 功能。

虽然 ext4 自身支持 project quota 功能（Linux4.5 版本之后）。但是，overlay over ext4 却不支持 project quota 功能。

经过我的尝试，采用 overlay over ext4 启动容器，会报以下错误：

Docker:Error response from daemon: --storage-opt is supported only for overlay over xfs with 'pquota' mount option.

官方文档里也有提到：For the overlay2 storage driver, the size option is only available if the backing fs is xfs and mounted with the pquota mount option（大意是 overlay 存储驱动仅支持基于 xfs 开启 project quota 功能）。

专栏中最有收获的文章是哪几篇?

首先是第 2 讲，我印象最深的是这篇文章最后的总结了，提到容器里 1 号进程对信号处理的两个要点：

1. 在容器中，1 号进程永远不会响应 SIGKILL 和 SIGSTOP 这两个特权信号；

2. 对于其他的信号，如果用户自己注册了 handler，1 号进程可以响应。

其中，要点 2 打破了我的固有认知。我原先一直认为在容器中 1 号进程无法被杀死，而这一讲结合相关内核源码与若干示例充分证明了这一要点，思路清晰，令人受益良多。

在第 9 讲学习 Memory Cgroup 的时候，课程里提到，每个容器的 Memory Cgroup 在统计每个控制组的内存使用时包含了两部分，RSS 和 Page Cache。

不过我认为，Momory Cgroup 除了包括 RSS 和 Page Cache，还包括了对内核内存的限制，作者给出的例子情况比较简单，基本没有使用 slab，倘若在容器中打开海量小文件，内核内存 inode、dentry 等会被计算在内。

内存使用量计算公式（memory.kmem.usage_in_bytes 表示该 memcg 内核内存使用量）：

memory.usage_in_bytes = memory.stat[rss] + memory.stat[cache] + memory.kmem.usage_in_bytes

另外，Memory Cgroup OOM 不是真正依据内存使用量 memory.usage_in_bytes，而是依据 working set（使用量减去非活跃 file-backed 内存），working set 计算公式是：

working_set=memory.usage_in_bytes-total_inactive_file。

接下来要说说第 14 讲，文章首先抛出一个引人关注的现象：应用程序从虚拟机迁移到容器并施加 Memory Cgroup 限制时，文件写入出现较大的延时波动。

我印象最深的是，这一讲通过程序复现、理论分析、工具追踪等一系列手段，最终得出一个令我深有同感的结论：在对容器做 Memory Cgroup 限制内存大小的时候，不仅要考虑容器中进程实际使用的内存量，还要考虑容器中程序 I/O 的量，合理预留足够的内存作为 Buffered I/O 的 Page Cache。

其实学习容器技术没不久，我就已经了解到，容器网络延迟相比宿主机高出 10% 左右，但并未深入思考过网络延迟变高的具体原因。

而专栏第 17 讲内容讲网络延时的时候，从 veth driver 的内核源码入手，合理解释了容器网络性能损失的来龙去脉，感觉收获很大，可以说是透过现象挖到了本质。受到这一讲的启发，对排查网络延迟问题，我又多了新的思路。

总结

在这个专栏学习结束后，我回过头来看，很高兴已经达到当初选择学习专栏的目的。

阅读过程整体比较顺畅，尤其觉得作者列出的 Linux 源码恰到好处，产生不少共鸣，因为在探究容器问题过程中，我也曾经阅读过这些代码。

感谢极客时间为我们提供了如此优秀的学习平台，也感谢作者为我们带来了非常精彩的容器实战课程。希望订阅这门课程的同学，也能有所收获，把学到的知识用到工作里。

我觉得，无论是学习容器，还是学习其他技术知识，就好像升级打怪一样，苦练基本功，手感和意识才会逐渐提升。最后，希望我们都能成为容器高手，让我们相信坚持的力量，终会厚积薄发！

提建议

上一篇 结课测试 | 这些容器技术的问题，你都掌握了么？

下一篇 加餐01 | 案例分析：怎么解决海量IPVS规则带来的网络延时抖动问题？

精选留言 (2)

写留言

CY

2021-02-10

感谢莫名同学！
几乎每一讲后面都可以看到你有深度的提问和评论。

1 3

我来也
2021-02-10

像学霸们学习！

你学的好过细啊，方法也很不错，学习了！
最后，文章也写的很好哦！

1