52 | 答疑：在问题中解决问题，在思考中产生思考

52 | 答疑：在问题中解决问题，在思考中产生思考

张磊 2018-12-21

13:11

讲述：张磊 大小：12.08M

在本篇文章中，我将会对本专栏部分文章最后的思考题，进行一次集中地汇总和答疑。希望能够帮助你更好地理解和掌握 Kubernetes 项目。

问题 1：你是否知道如何修复容器中的 top 指令以及 /proc 文件系统中的信息呢？（提示：lxcfs）

其实，这个问题的答案在提示里其实已经给出了，即 lxcfs 方案。通过 lxcfs，你可以把宿主机的 /var/lib/lxcfs/proc 文件系统挂载到 Docker 容器的 /proc 目录下。使得容器中进程读取相应文件内容时，实际上会从容器对应的 Cgroups 中读取正确的资源限制。 从而得到正确的 top 命令的返回值。

> *问题选自第 6 篇文章《白话容器基础（二）：隔离与限制》。*

问题 2：既然容器的 rootfs（比如，Ubuntu 镜像），是以只读方式挂载的，那么又如何在容器里修改 Ubuntu 镜像的内容呢？（提示：Copy-on-Write）

这个问题的答案也同样出现在了提示里。

简单地说，修改一个镜像里的文件的时候，联合文件系统首先会从上到下在各个层中查找有没有目标文件。如果找到，就把这个文件复制到可读写层进行修改。这个修改的结果会屏蔽掉下层的文件，这种方式就被称为 copy-on-write。

> *问题选自第 7 篇文章《白话容器基础（三）：深入理解容器镜像》。*

问题 3：你在查看 Docker 容器的 Namespace 时，是否注意到有一个叫 cgroup 的 Namespace？它是 Linux 4.6 之后新增加的一个 Namespace，你知道它的作用吗？

Linux 内核从 4.6 开始，支持了一个新的 Namespace 叫作：Cgroup Namespace。 我们知道，正常情况下，在一个容器里查看 /proc/$PID/cgroup，是会看到整个宿主机的 cgroup 信息的。而有了 Cgroup Namespace 后，每个容器里的进程都会有自己 Cgroup Namespace，从而获得一个属于自己的 Cgroups 文件目录视图。也就是说，Cgroups 文件系统也可以被 Namespace 隔离起来了。

> *问题选自第 8 篇文章《 白话容器基础（四）：重新认识 Docker 容器》。*

问题 4：你能否说出，Kubernetes 使用的这个“控制器模式”，跟我们平常所说的“事件驱动”，有什么区别和联系吗？

这里“控制器模式”和“事件驱动”最关键的区别在于：

> 对于控制器来说，被监听对象的变化是一个持续的信号，比如变成 ADD 状态。只要这个状态没变化，那么此后无论任何时候控制器再去查询对象的状态，都应该是 ADD。
> 而对于事件驱动来说，它只会在 ADD 事件发生的时候发出一个事件。如果控制器错过了这个事件，那么它就有可能再也没办法知道 ADD 这个事件的发生了。

> *问题选自第 16 篇文章《编排其实很简单：谈谈“控制器”模型》。*

问题 5：在实际场景中，有一些分布式应用的集群是这么工作的：当一个新节点加入到集群时，或者老节点被迁移后重建时，这个节点可以从主节点或者其他从节点那里同步到自己所需要的数据。

在这种情况下，你认为是否还有必要将这个节点 Pod 与它的 PV 进行一对一绑定呢？（提示：这个问题的答案根据不同的项目是不同的。关键在于，重建后的节点进行数据恢复和同步的时候，是不是一定需要原先它写在本地磁盘里的数据）

这个问题的答案是不需要。

像这种不依赖于 PV 保持存储状态或者不依赖于 DNS 名字保持拓扑状态的”非典型“应用的管理，都应该使用 Operator 来实现。

> *问题选自第 19 篇文章《深入理解 StatefulSet（二）：存储状态》。*

问题 6：我在文中提到，在 Kubernetes v1.11 之前，DaemonSet 所管理的 Pod 的调度过程，实际上都是由 DaemonSet Controller 自己而不是由调度器完成的。你能说出这其中有哪些原因吗？

这里的原因在于，默认调度器之前的功能不是很完善，比如，缺乏优先级和抢占机制。所以，它没办法保证 DaemonSet ，尤其是部署时候的系统级的、高优先级的 DaemonSet 一定会调度成功。这种情况下，就会影响到集群的部署了。

> *问题选自第 21 篇文章《容器化守护进程的意义：DaemonSet》。*

问题 7：在 Operator 的实现过程中，我们再一次用到了 CRD。可是，你一定要明白，CRD 并不是万能的，它有很多场景不适用，还有性能瓶颈。你能列举出一些不适用 CRD 的场景么？你知道造成 CRD 性能瓶颈的原因主要在哪里么？

CRD 目前不支持 protobuf，当 API Object 数量 >1K，或者单个对象 >1KB，或者高频请求时，CRD 的响应都会有问题。 所以，CRD 千万不能也不应该被当作数据库使用。

其实像 Kubernetes ，或者说 Etcd 本身，最佳的使用场景就是作为配置管理的依赖。此外，如果业务需求不能用 CRD 进行建模的时候，比如，需要等待 API 最终返回，或者需要检查 API 的返回值，也是不能用 CRD 的。同时，当你需要完整的 APIServer 而不是只关心 API 对象的时候，请使用 API Aggregator。

> *问题选自第 27 篇文章《聪明的微创新：Operator 工作原理解读》。*

问题 8：正是由于需要使用“延迟绑定”这个特性，Local Persistent Volume 目前还不能支持 Dynamic Provisioning。你是否能说出，为什么“延迟绑定”会跟 Dynamic Provisioning 有冲突呢？

延迟绑定将 Volume Bind 的时机，推迟到了第一个使用该 Volume 的 Pod 到达调度器的时候。可是对于 Dynamic Provisioning 来说，它是要在管理 Volume

的控制循环里就为 PVC 创建 PV 然后绑定起来的，这个时间点跟 Pod 被调度的时间点是不相关的。

> *问题选自第 29 篇文章《 PV、PVC 体系是不是多此一举？从本地持久化卷谈起》。*

问题 9：请你根据编写 FlexVolume 和 CSI 插件的流程，分析一下什么时候该使用 FlexVolume，什么时候应该使用 CSI？

在文章中我其实已经提到，CSI 与 FlexVolume 的最大区别，在于 CSI 可以实现 Provision 阶段。所以说，对于不需要 Provision 的情况，比如你的远程存储服务总是事先准备好或者准备起来非常简单的情况下，就可以考虑使用 FlexVolume。但在生产环境下，我都会优先推荐 CSI 的方案。

> *问题选自第 31 篇文章《容器存储实践：CSI 插件编写指南》。*

问题 10：Flannel 通过“隧道”机制，实现了容器之间三层网络（IP 地址）的连通性。但是，根据这个机制的工作原理，你认为 Flannel 能保证容器二层网络（MAC 地址）的连通性吗？为什么呢？

不能保证，因为“隧道”机制只能保证被封装的 IP 包可以到达目的地。而只要网络插件能满足 Kubernetes 网络的三个假设，Kubernetes 并不关心你的网络插件的实现方式是把容器二层连通的，还是三层连通的。

> *问题选自第 33 篇文章《深入解析容器跨主机网络》。*

问题 11：你能否能总结一下三层网络方案和“隧道模式”的异同，以及各自的优缺点？

在第 35 篇文章的正文里，我已经为你讲解过，隧道模式最大的特点，在于需要通过某种方式比如 UDP 或者 VXLAN 来对原始的容器间通信的网络包进行封装，然后伪装成宿主机间的网络通信来完成容器跨主通信。这个过程中就不可避免的需要封包和解封包。这两个操作的性能损耗都是非常明显的。而三层网络方案则避免了这个过程，所以性能会得到很大的提升。

不过，隧道模式的优点在于，它依赖的底层原理非常直白，内核里的实现也非常成熟和稳定。而三层网络方案，相对来说维护成本会比较高，容易碰到路由规则分发和设置出现问题的情况，并且当容器数量很多时，宿主机上的路由规则会非常复杂，难以 Debug。

所以最终选择选择哪种方案，还是要看自己的具体需求。

> *问题选自第 35 篇文章《解读 Kubernetes 三层网络方案》。*

问题 12：为什么宿主机进入 MemoryPressure 或者 DiskPressure 状态后，新的 Pod 就不会被调度到这台宿主机上呢？

在 Kubernetes 里，实际上有一种叫作 Taint Nodes by Condition 的机制，即当

Node 本身进入异常状态的时候，比如 Condition 变成了 DiskPressure。那么，Kubernetes 会通过 Controller 自动给 Node 加上对应的 Taint，从而阻止新的 Pod 调度到这台宿主机上。

> *问题选自第 40 篇文章《Kubernetes 的资源模型与资源管理》。*

问题 13：Kubernetes 默认调度器与 Mesos 的“两级”调度器，有什么异同呢？

Mesos 的两级调度器的设计，是 Mesos 自己充当 0 层调度器（Layer 0），负责统一管理整个集群的资源情况，把可用资源以 Resource Offer 的方式暴露出去；而上层的大数据框架（比如 Spark）则充当 1 层调度器（Layer 1），它会负责根据 Layer 0 发来的 Resource Offer 来决定把任务调度到某个具体的节点上。这样做的好处是：

第一，上层大数据框架本身往往自己已经实现了调度逻辑，这样它就可以很方便地接入到 Mesos 里面；
第二，这样的设计，使得 Mesos 本身能够统一地对上层所有框架进行资源分配，资源利用率和调度效率就可以得到很好的保证了。

相比之下，Kubernetes 的默认调度器实际上无论从功能还是性能上都要简单得多。这也是为什么把 Spark 这样本身就具有调度能力的框架接入到 Kubernetes 里还是比较困难的。

> *问题选自第 41 篇文章《十字路口上的 Kubernetes 默认调度器》。*

问题 14：当整个集群发生可能会影响调度结果的变化（比如，添加或者更新 Node，添加和更新 PV、Service 等）时，调度器会执行一个被称为 MoveAllToActiveQueue 的操作，把所调度失败的 Pod 从 unscheduelableQ 移动到 activeQ 里面。请问这是为什么？

一个相似的问题是，当一个已经调度成功的 Pod 被更新时，调度器则会将 unschedulableQ 里所有跟这个 Pod 有 Affinity/Anti-affinity 关系的 Pod，移动到 activeQ 里面。请问这又是为什么呢？

其实，这两个问题的答案是一样的。

在正常情况下，默认调度器在调度失败后，就会把该 Pod 放到 unschedulableQ 里。unschedulableQ 里的 Pod 是不会出现在下个调度周期里的。但是，当集群

本身发生变化时，这个 Pod 就有可能再次变成可调度的了，所以这时候调度器要把它们移动到 activeQ 里面，这样它们就获得了下一次调度的机会。

类似地，当原本已经调度成功的 Pod 被更新后，也有可能触发 unschedulableQ 里与它有 Affinity 或者 Anti-Affinity 关系的 Pod 变成可调度的，所以它也需要获得“重新做人”的机会。

> *问题选自第 43 篇文章《Kubernetes 默认调度器的优先级与抢占机制》。*

问题 15：请你思考一下，我前面讲解过的 Device Plugin 为容器分配的 GPU 信息，是通过 CRI 的哪个接口传递给 dockershim，最后交给 Docker API 的呢？

既然 GPU 是 Devices 信息，那当然是通过 CRI 的 CreateContainerRequest 接口。这个接口的参数 ContainerConfig 里就有容器 Devices 的描述。

> *问题选自第 46 篇文章《解读 CRI 与 容器运行时》。*

问题 16：安全容器的意义，绝不仅仅止于安全。你可以想象一下这样一个场景：比如，你的宿主机的 Linux 内核版本是 3.6，但是应用却必须要求 Linux 内核版本是 4.0。这时候，你就可以把这个应用运行在一个 KataContainers 里。那么请问，你觉得使用 gVisor 是否也能提供这种能力呢？原因是什么呢？

答案是不能。gVisor 的实现里并没有一个真正的 Linux Guest Kernel 在运行。所以它不能像 KataContainers 或者虚拟机那样，实现容器和宿主机不同 Kernel 甚至不同操作系统的需求。

但还是要强调一下，以 gVisor 为代表的用户态 Kernel 方案是安全容器的未来，只是现在还不够完善。

> *问题选自第 47 篇文章《绝不仅仅是安全：Kata Containers 与 gVisor》。*

问题 17：将日志直接输出到 stdout 和 stderr，有没有什么其他的隐患或者问题呢？如何进行处理呢？

这样做有一个问题，就是日志都需要经过 Docker Daemon 的处理才会写到宿主机磁盘上，所以宿主机没办法以容器为单位进行日志文件的 Rotate。这时候，还是要考虑通过宿主机的 Agent 来对容器日志进行处理和收集的方案。

> *问题选自第 50 篇文章《让日志无处可逃：容器日志收集与管理》。*

问题 18：你能说出 Kubernetes 社区同 OpenStack 社区相比的不同点吗？你觉得各有哪些优缺点呢？

OpenStack 社区非常强调民主化，治理方式相对松散，这导致它在治理上没能把主线和旁线分开，政治和技术也没有隔离。这使得后期大量的低价值或者周边型的项目不断冲进 OpenStack 社区，大大降低了社区的含金量，并且分散了大量的社区精力在这些价值相对不高的项目上，从而拖慢并干扰了比如 Cinder、Neutron 等核心项目的演进步伐和方向，最终使得整个社区在容器的热潮下难以掉头，不可避免地走向了下滑的态势。

相比之下，CNCF 基金会成功地帮助 Kubernetes 社区分流了低价值以及周边型项目的干扰，并且完全承接了 Marketing 的角色，使得 Kubernetes 社区在后面大量玩家涌入的时候，依然能够专注在主线的演进上。

Kubernetes 社区和 OpenStack 社区的这个区别，是非常关键的。

> *问题选自第 51 篇文章《谈谈 Kubernetes 开源社区和未来走向》。*