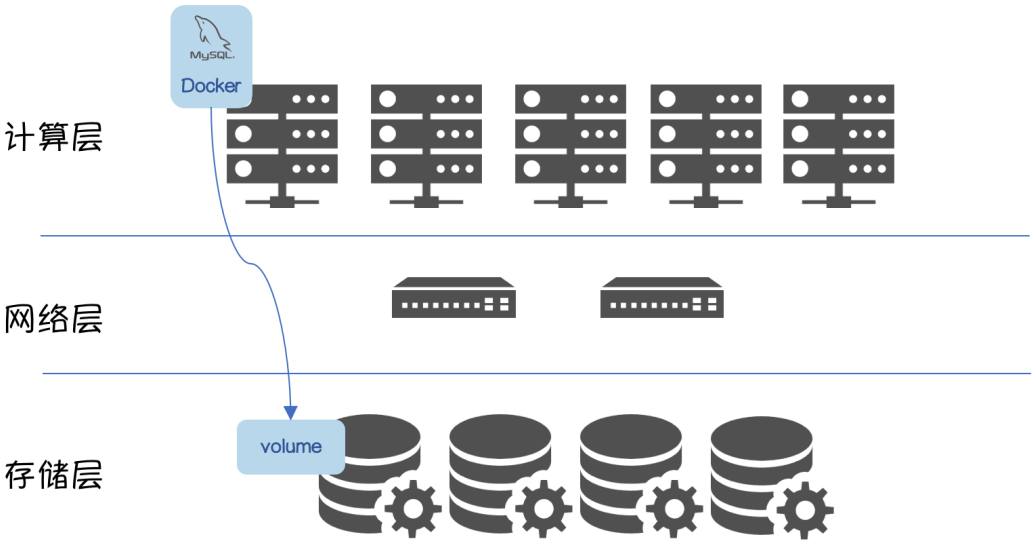
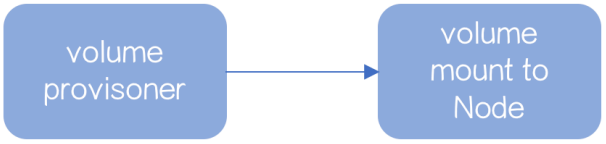
即使不使用 Kubernetes, 在编排持久化 workload 时, **你都需要了解, 编排框架和 CloudProvider 是如何交互的, 数据又是如何被写"坏"的**. 以下描述的场景具有普遍意义, 也是必须要回答的问题.

简单场景

- 从存储池中获取 RW Volume, 挂载到指定 Node 上, 并在该 Node 上启动持久化应用 MySQL



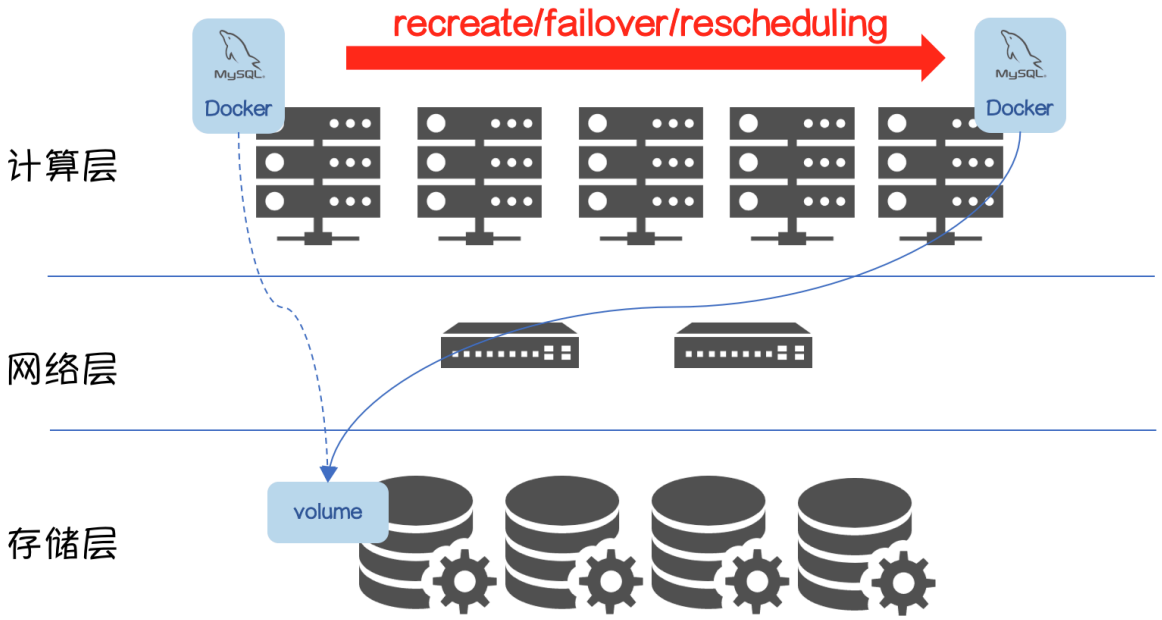
Volume 使用流程大致如下:



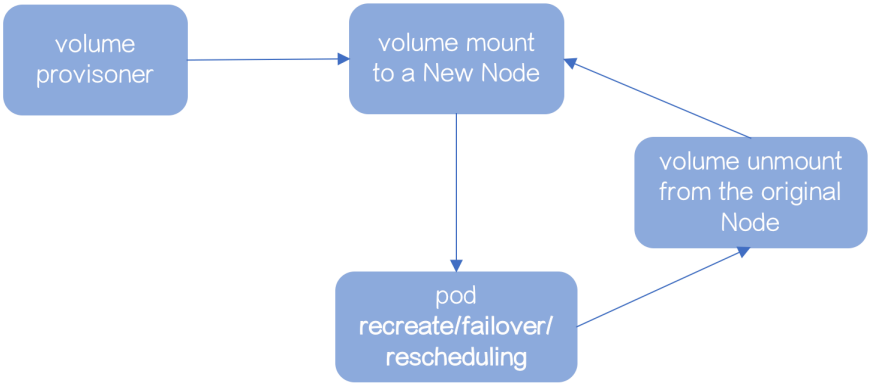
- 生成 Volume
- mount 到数据库实例所在节点,数据库启动

再复杂一点

- 从存储池中获取 Volume, 挂载到指定 Node 上, 并在该 Node 上启动持久化应用 MySQL
- MySQL具备重建/故障切换/重新调度的能力



Volume 使用流程也会更复杂一些：



- 生成 Volume
- mount 到数据库实例所在节点,数据库启动
- 数据库实例因为recreate/failover/rescheduling, 被调度新节点
- Volume 从原节点 unmount
- mount 到新节点,数据库启动

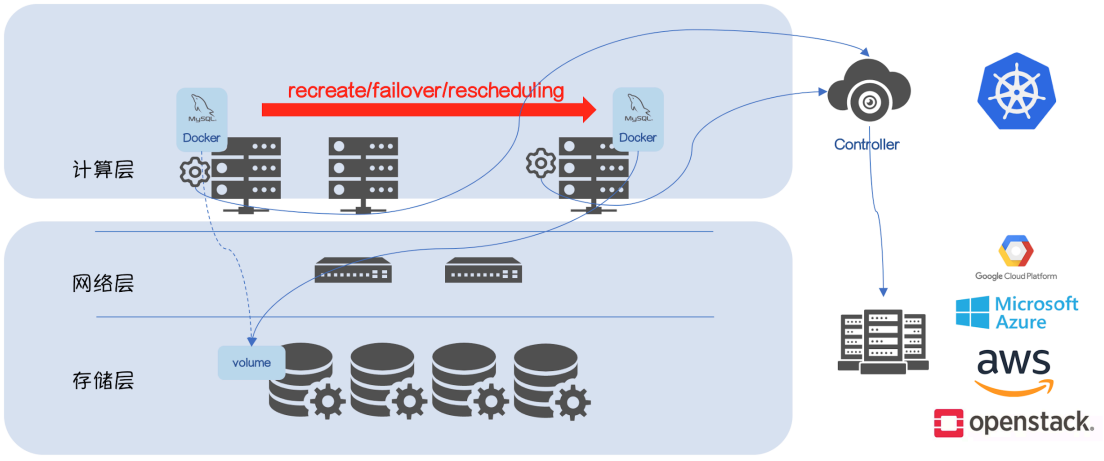
相比上个场景, 增加了3个环节.

再复杂一点

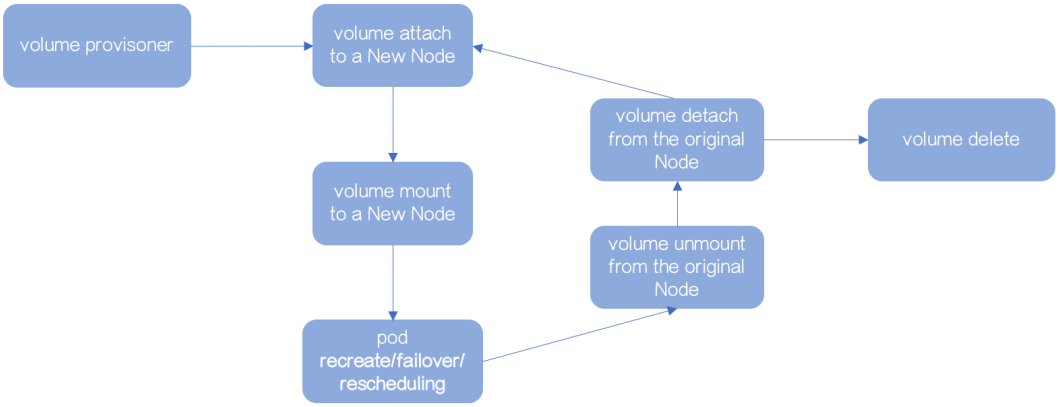
大多数情况下 Kubernetes 不会直接管理 bare-metal, 而是运行在第三方 CloudProvider 上 (*GCE/Azure/AWS/OpenStack*), Kubernetes 会作为 Volume 的使用者, 由 CloudProvider 负责 Volume 的生命周期, 所以之前的 mount/unmount 会有所变化：

- Volume 在 mount 之前, 需要”通知” CloudProvider
- Volume 在 unmount 之后, 需要”通知” CloudProvider

如果Volume 在 unmount 之后, 没有”通知” CloudProvider, CloudProvider 会保证该 Volume 不会被挂载到其他 Node 上, **“多点挂载”在大多数场景下会导致”Data Corruption”**, 所以添加这两个步骤是有必要的, CloudProvider 需要感知 Volume 的”使用场景”(譬如在 GCE 环境, 是不允许RW Volume 同时挂载到多个节点). 这两个步骤被称为 attach/detach.



Volume 使用流程也会更复杂一些：



- 生成 Volume
- attach 到数据库实例所在节点
- mount 到数据库实例所在节点,数据库启动
- 数据库实例因为recreate/failover/rescheduling, 被调度其他节点
- Volume 从原节点 unmount
- Volume 从原节点 detach

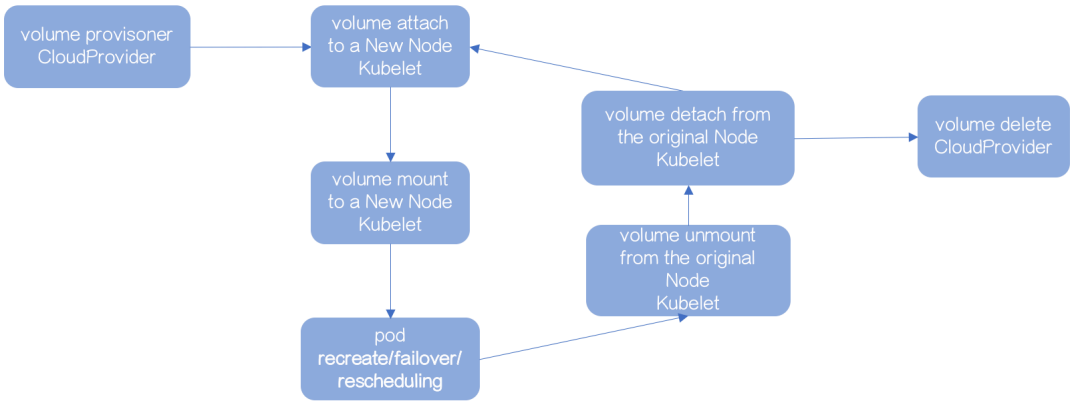
- attach 到新节点
- mount 到新节点,数据库启动

再复杂一点

需要继续思考一个问题：

谁来”通知”CloudProvider?

Kubernetes 1.3 之前, 以上所有的工作由 Kubelet 完成, 由Volume Plugin 适配第三方 CloudProvider 的逻辑[1].



但 Kubelet 是运行在 Node 端的 agent

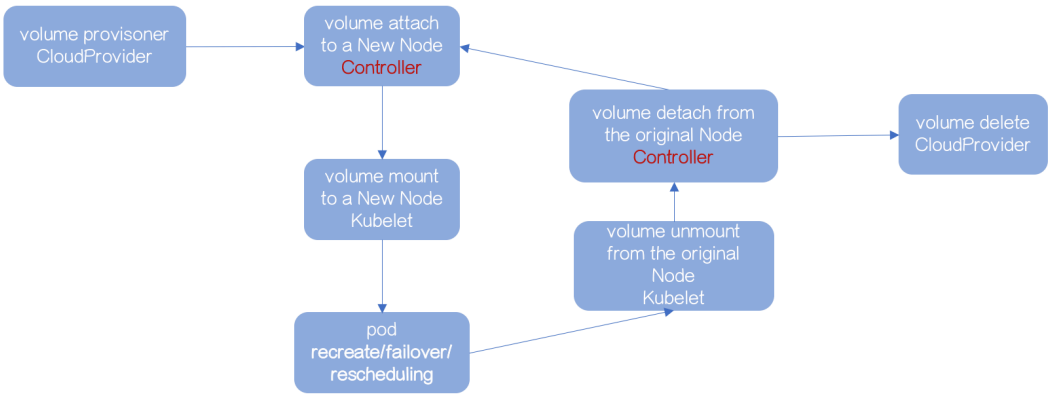
一旦 Node 重启 / Crash / 网络故障, 都会导致无法”通知”CloudProvider, 即便该 Volume 已经没有应用访问, CloudProvider 都不会让任何节点使用它.

当然, 还会有其他问题, 譬如多个 Kubelet 带来的 ”race condition”.

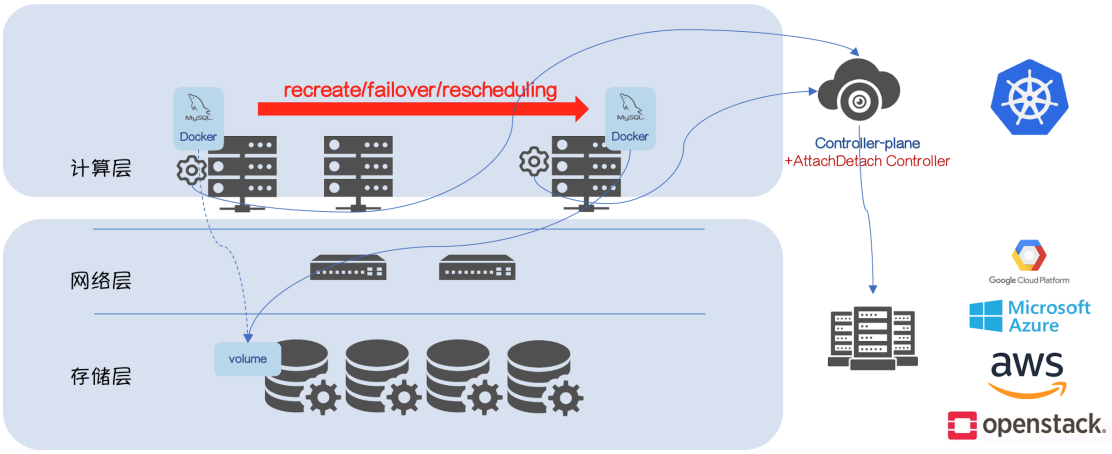
解耦 Attach-Mount-Unmount-Detach

流程不变, Kubernetes 1.3 之后, 尝试使用专门的 Controller 管理 Attach 和 Detach 操作.

[1] more details :
https://github.com/kubernetes/kubernetes/blob/7bd2263566d434869ff501f2fb824373fcdf4e03/cmd/kubelet/app/plugins.go#L68



该 Controller 被叫做 AttachDetach Controller, 它运行在已有的 Controller Plane 上



通过"volumes.kubernetes.io/controller-managed-attach-detach"[2]启动该特性(默认使用该特性).

问题还没有解决

attach-mount-umount-detach流程的串行有序是保障数据不被写"坏"的基础.

- Volume 在 mount 之前, Kubelet 会先确认是否已经 attach[3]
- Volume 在 detach 之前, AttachDetach Controller 会确认是否已经 unmount[4]

---

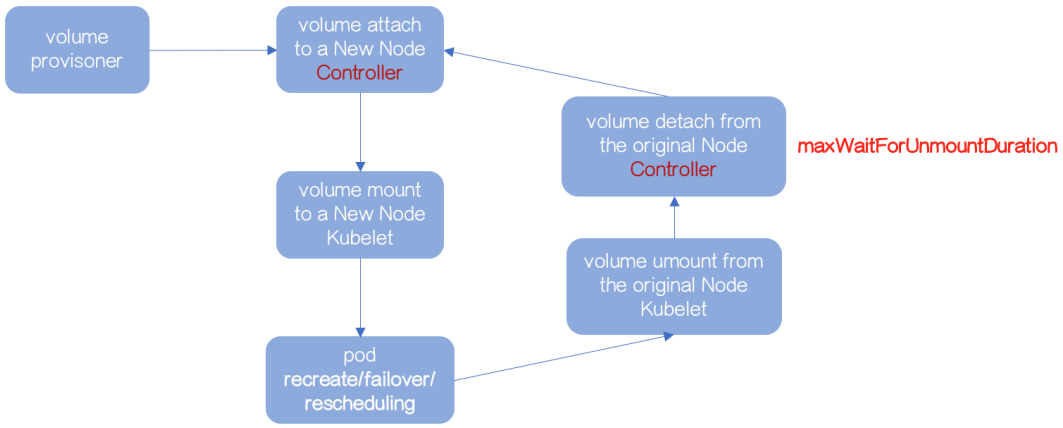
[2] more details :
https://github.com/kubernetes/kubernetes/blob/9847c8ee0a2c71fc4e9e1665017f05de7c9f815e/pkg/volume/util/util.go#L65
[3] more details :
https://github.com/kubernetes/kubernetes/blob/f4e71f1c748c00f424548b57e1d7694ff11d6139/pkg/volume/util/operationexecutor/operation_generator.go#L1114
[4] more details :
https://github.com/kubernetes/kubernetes/blob/f4e71f1c748c00f424548b57e1d7694ff11d6139/pkg/volume/util/operationexecutor/operation_generator.go#L404

所以, 如果 Volume 不能被 Kubelet 成功地 unmount, AttachDetach Controller 不能进行 detach 操作.

又回到之前的问题

Kubelet 是运行在 Node 端的 agent , 一旦 Node 重启 / Crash / 网络故障, 都会导致无法完成 unmount 操作.

AttachDetach Controller 不可能无限制的等待前置动作 unmount, 所以通过参数 maxWaitForUnmountDuration[5] (默认6分钟)解决该问题.



超过 maxWaitForUnmountDuration, **AttachDetach Controller 会启动 force detaching[6].**

这破坏了 attach-mount-umount-detach流程的串行有序[7], 一个 RW Volume 在多个节点上挂载的可能性出现了[8].

数据可能被写"坏"

Kuberetes 集群的正常运行, 依赖 API Server 跟 Kubelet 的正常交互, 可以理解为”心跳”.

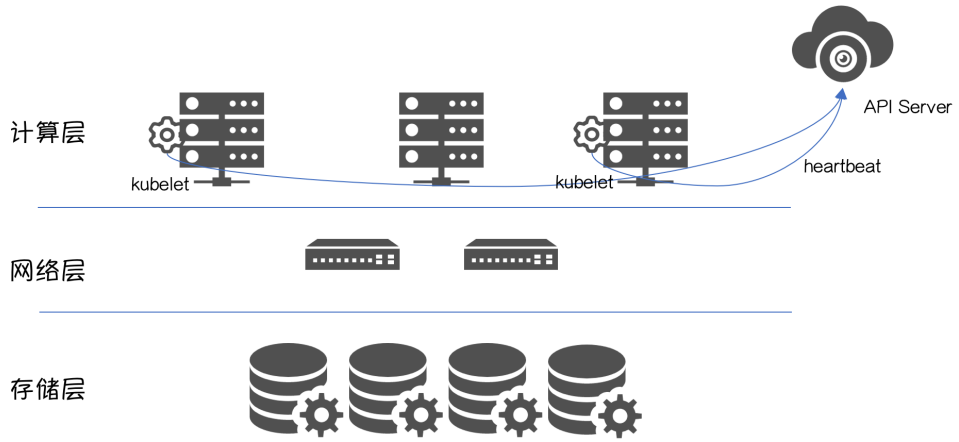
---

[5] more datails : https://github.com/kubernetes/kubernetes/blob/317853c90c674920bfbbdac54fe66092ddc9f15f/pkg/controller/volume/attachdetach/reconciler/reconciler.go#L64

[6] more details : https://github.com/kubernetes/kubernetes/blob/317853c90c674920bfbbdac54fe66092ddc9f15f/pkg/controller/volume/attachdetach/reconciler/reconciler.go#L229

[7] 为了保证有序, AttachDetach Controller 和 Volume Manager 中加入了大量前置判断的代码

[8] 又叫”多点挂载”



”心跳”丢失的可能性很多, 譬如:

1. Node 重启 / Crash;
2. **Node 跟 API Server 网络故障;**
3. **Node 在高负载下, Kubelet无法获得 CPU 时间分片;**
4. 等等

换句话说, 一旦”心跳”丢失,集群无法判断 **Node 的真实状态**. 这时运行在 Controller Plane 之上的 NodeLifecycle Controller 会把该节点标记为”**ConditionUnknown**”.

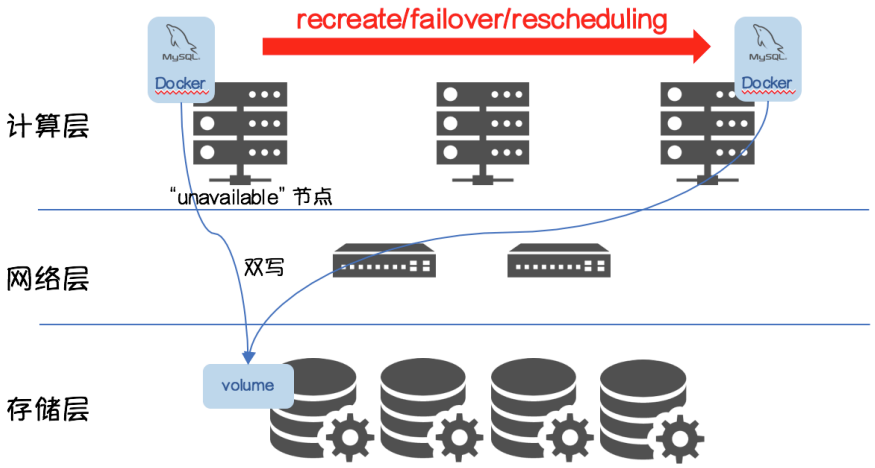
一旦超过阈值podEvictionTimeout[9], NodeLifecycle Controller会对该节点上运行的 MySQL 进行驱逐[10], Scheduler会将 MySQL调度到其他”available”节点.

配合上 **force detaching 导致的”多点挂载”,** 多个实例对同一个 Volume 的 “Write”[11] 导致” Data Corruption”.

[9] 通过pod-eviction-timeout传入

[10] 当”心跳”丢失, Kubelet 并不能接收到 NodeLifecycle Controller发起的”驱逐”, 该”驱逐”是仅仅是 etcd 中的数据字典. 好比一个人失踪了 20 年, 大家把他标记为”死亡”, 但是真实的他可能还好好的活着.

[11] 又叫”双写”



总结

Kubernetes 是极好的编排平台, 前提是我们需要深入的了解她.