# **快照的回滚**

1. **对外新增API**

### RC 结构体新增 rollbackTo 和 rollbackStatus

|  |
| --- |
| {  "kind": "ReplicationController",  "metadata": {  "name": "testrprc",  "namespace": "09c06d05f36043938326094a89145a2d",  },  **"spec": {**  **"rollbackTo": {**  **"snapshotName" : "",**  **"dataHoldPVName" : ""**  **}**  **},**  **"status": {**  **"rollbackStatus": {**  **"phase":"",**  **"message":""**  **}**  **},**  .....  } |

### PUT

|  |
| --- |
| /api/v1/namespaces/{namespace}/replicationcontroller/{name}/rollback |

里传入的 body 里的 snapshotName 和 dataHoldPVName 必须有且仅有一个非空，否则直接返回 400 错误

1. **实现原理**

1）上层先 GET 用户当前需要回滚容器RC 对应的真正使用的 dataHoldPVName，保存起来（后面反回滚时用，如果成功需要删除该PV）

2）上层 调用 PUT PUT /api/v1/namespaces/{namespace}/replicationcontroller/{name}/rollback 发起回滚请求成功

3）RC 控制器感知到 Update 事件时，判断 rollbackTo 是否指针非空（且仅有一个非空）

如果快照回滚（必然 snapshotName 非空） 则把该 rc 放入新加的一个单独的 FIFO 队列（ 默认是多优先级队列）（Update）

增加单独的FIFO是为了避免影响正常的RC 创建 pod/pv/netEndpoint 的流程，对应的专用工作线程数跟先保持 rc 一致

4）RC 控制器的多单优先级取出需要回滚的 RC 对象根据里面的 snapshotName 创建一个 pv 资源（检查 snapshotName 是否在k8s 里确实存在有效，否则直接更新 rc 的 rollbackStatus phase=Failed, message注明失败原因）

pv 控制器会负责将 snapshot 创建 云硬盘卷

然后该RC的工作协程里不断轮询 pv 的状态（sleep 200ms 左右），直到PV的状态变成 Available 或者 Failed 为止。（如果状态一直 pending 怎么办？？？）

5）PV 控制器

pv 控制器 感知到 Add/Update 后使用 其中的快照创建一个新的 pv并不断后台检查是否 pv对应的卷变成 Avialble（管理平台提供一个最大时间，超过了就直接回滚失败）如果是的，则先更新 rc 的 rollbackStatus 再更新 pv 的 status 为 Avialble。 【成功是否可以不更新 rc 的 rollbackStatus】

6）RC 控制器发现 PV 变成 Failed ，则修改 RC 的 rollbackStatus=Failed, message 注明失败原因RC 控制器发现 PV 变成 Available 后做真正的回滚操作具体步骤是先检查当前关联的 pod 使用的pv 是不是需要回滚的 pv不是则修改 rc.template 的 外网netEndPonit, 新快照创建的卷, 老podNmae, nodeName 也修改下（尽可能copy 原来的pod内容） 开始做真正的回滚

a> 删除老的 pod ，删除前先check 是否需要删除 外网ip, 和当前正在使用的 pv. 【 nodeController 的 deletePod 的逻辑做一遍： 包括 node free, detach 卷】目前的回滚需要保留外网ip, 外网ip不能删， pv 因为回滚可能失败 所以pv现在也还不能删（ 上层读到 pv 不对怎么办？？？）【删除 pod 的时候 pv 只需要 detach 变为自由状态即可， pv 资源本身先不需要主动删除，最后留给上层 pv 删除 】

b> 删除pod 后 RC 控制器感知到 pod 的删除事件，会转化为 RC 的update 行为: 给 rc 队列加一个 Update Rc 的任务放到单独的回滚 FIFO 工作队列（Update）

c> RC 回滚的工作线程取出任务，用新的快照 PV 创建新的 Pod， 外网Ip 和 podName, nodeName 都用老的，不需要重新调度（依然使用老的 node）

【A: 直接更新 pod 资源？ 】

【B：删除老 Pod 时，立马串行直接创建新的 （etcd 有风险：可能处理反过来了） watch 可能被合并遗漏吗？】 后台检查rc回滚work FIFO 工作队列的发现 doing 不会执行？这边成功了那边最多多检查一下而已。 如果中间重启就只能通过模板恢复了！！

【C：删除就好了， 创建统一靠 回滚的 work FIFO 通过模板创建 】

删除老的net/pv/pod

用新的 net/pv/ pod

最好直接 kubelet 本地能根据 pod 与 pv 的关联关系 自动更新 pv

最差情况是： 只删除老pod，创建新pod 新老pod的区别仅仅是 pv 【刚删除老pod，又把 pod 新建出来了 kubelet 能正确处理吗？ 能做优化吗？比如外网不用重新初始化。 上层还需要初始化私有网ip】

修改 rc 的 replica = 0 的方式删除老 pod, 同时把老的外网netEndPonit 和新快照创建的卷及老podNmae 更新到 rc.template 里也一并修改了（这个保证删除老pod并确保使用新的pv和老的ip的原子性的唯一方式）修改完成，这个RC的任务必须临时结束（让另外的线程将老pod删除，见下面步骤7 ） [k8s maxWaitTime]

7） kubelet 感知到 老的pod 被删除了

stop pod

卸载外网ip

umount 老的目录

卸载老的卷

8） kubelet感知到 新加了 pod (仅仅pv不同)

挂载外网ip

挂载新的卷

mount 卷到的老目录

启动容器

9） 上层需要管理平台通知用户等待时间（但是由 rc 的rollbackStatus 和 pod running 的状态决定最后的成功还是失败）

10）上层check 到新的容器 running 起来了， 删除老的 pv， 最后初始化新的容器的私有网ip