数据分离设计文档

## 修订记录

|  |  |  |  |
| --- | --- | --- | --- |
| 修订版本号. | 修订内容简述 | 修订日期 | 修订人 |
| 1.0 | 创建文档 | 2016.3.7 | 方应 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 概述

Docker容器的设计原理导致docker从一个镜像重新创建后，原来的运行时数据将会丢失。本功能旨在保持docker容器运行过程中的数据并在用户正式删除容器前保证用户运行数据不会丢失。并在发生故障时能够进行迁移，并恢复容器数据。

## 需求说明

### 需求概述

目前所有的容器都运行在本地盘上，不能满足用户对容量的需求，同时对后续的容器迁移功能也会受到限制。即需要提供更大的盘为用户提供服务，同时也能为后续的产品功能提供更强大的功能。

目前运行的模式如下：

### 功能需求

目标效果：

数据保持：

|  |  |
| --- | --- |
| 编号 | 描述 |
| K8s-sjfl-fun-1 | 支持将容器数据和volume数据保持（默认云硬盘） |
| K8s-sjfl-fun-2 | 支持将数据保持到本地盘和云硬盘两种模式 |
| K8s-sjfl-fun-3 | 支持将用户-v指定的数据盘按照默认盘处理 |
| K8s-sjfl-fun-4 | 支持容器重启后自动从老数据恢复 |
| K8s-sjfl-fun-5 | 数据保持盘支持垂直扩容 |
| K8s-sjfl-fun-6 | 支持挂载额外的数据盘进行数据备份 |

快照：

|  |  |
| --- | --- |
| 编号 | 描述 |
| K8s-kz-fun-1 | 对运行的容器直接打快照 |
| K8s-kz-fun-2 | 支持从快照恢复容器 |
| K8s-kz-fun-3 | 删除容器时自动打快照 |

### 非功能需求

1. 性能

数据保持：

|  |  |
| --- | --- |
| 编号 | 描述 |
| K8s-sjfl-pf-1 | 容器创建流程在容器创建基础上加上云硬盘操作时间左右合理范围（指标暂时没有确定） |

1. 可用性/可靠性
2. 可扩展性
3. 可维护性

数据保持：

|  |  |
| --- | --- |
| 编号 | 描述 |
| K8s-sjfl-mt-1 | 容器数据保持的云硬盘可维护 |

## 相关术语

Pv和pvc

<http://kubernetes.io/docs/user-guide/persistent-volumes/>

<https://github.com/kubernetes/kubernetes.github.io/blob/master/docs/user-guide/persistent-volumes/index.md>

## 参考文档

[\\10.240.155.95\nce\](file:///\\10.240.155.95\nce\)【云计算】2期\需求\NCE\\快照与数据盘分离场景说明.docx

# 功能总体设计

## 总体设计概述

数据保持：

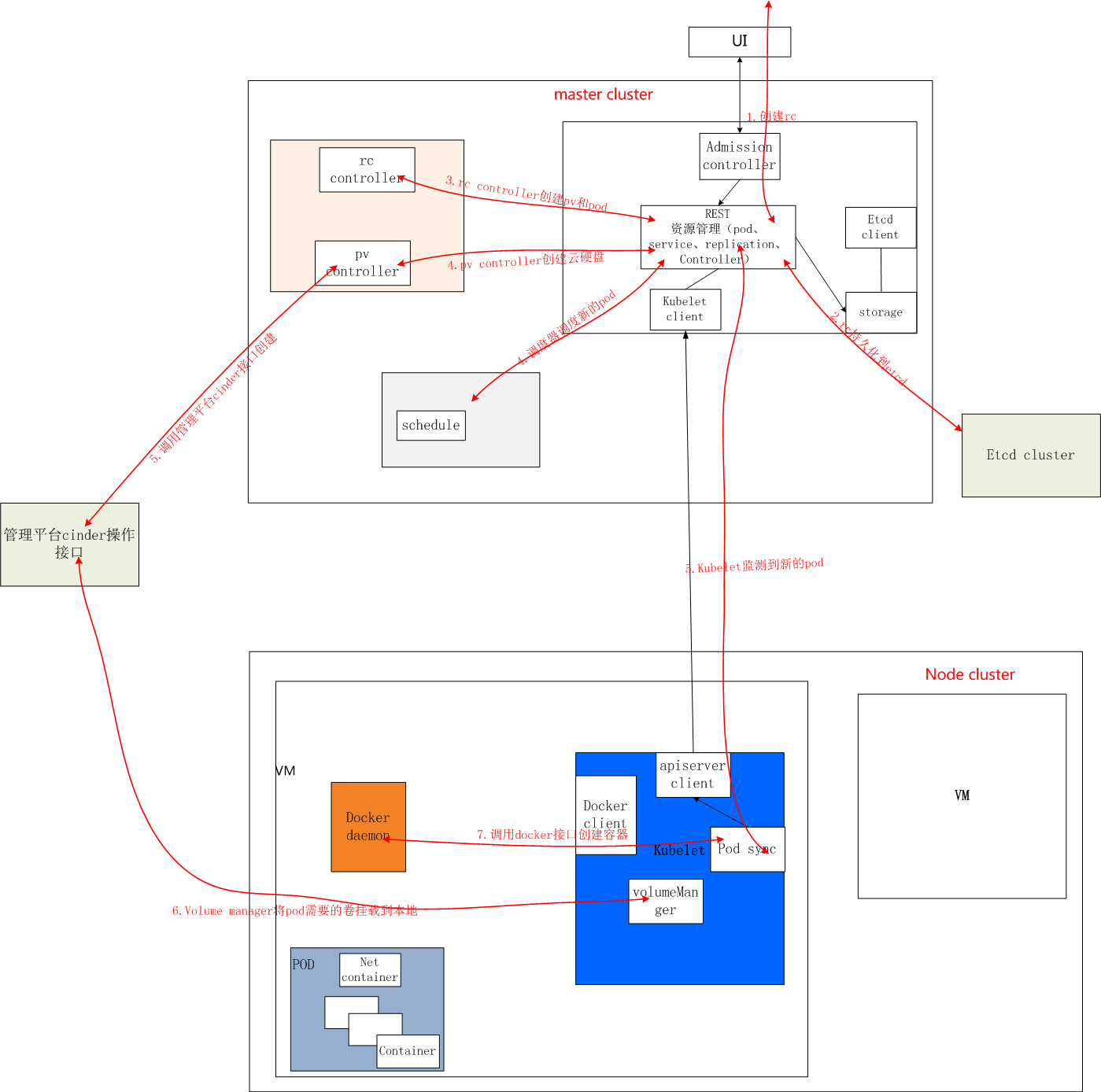


图2-1 总体设计图

创建rc的大致流程和涉及的模块如图2-1所示。rc controller负责在创建pod的时候为pod创建一个默认的云硬盘描述pv，创建pod描述并将两者关联。pv controller负责根据pv描述调用管理平台的相关cinder接口创建云硬盘，这其中也包括了种种权限认证。与创建pv并行的是调度器的调度。调度完成之后，node节点的kubelet会检测到新的pod需要创建，创建pod相关容器前，kubelet会首先检测pv关联的云硬盘是否创建成功，如果成功则调用相关的volume接口挂载pv描述的云硬盘到node节点上，失败则进入重试流程。云硬盘挂载成功，kubelet调用docker接口进行创建容器。

快照功能：

根据迭代情况决定是否在本迭代完成。

## 涉及模块功能设计描述

### 数据保持

Controller：

控制器涉及到rc congroller、pv congroller和node controller。

rc controller需要在创建pod时创建默认的云硬盘需求描述资源pv，并将其和pod关联。当然也需要根据rc配置的数据保持策略决定采用本地保持还是云硬盘。（rc.spec.startPolicy= "UseOld"使用本地盘，rc.spec.startPolicy= "ScheduleOld"则会使用云硬盘）

pv controller需要在检测到新的pv描述时调用管理平台的cinder相关接口创建云硬盘。并更新pv描述（设置volumeID等）。

node controller需要在node节点失联超过设定值后根据pod配置的数据保持策略进行区分处理，如果pod配置为保持数据到本地存储，node controller需要忽略该node，并产生报警。如果pod配置为保持数据到云硬盘，node controller需要解除pod和node的绑定关系，调度器将会重新介入调度。

Scheduler：

调度器需要修改调度策略，如果rc选择了本地数据保持，调度完成之后需要对node打上pod占用标签。

Kubelet：

需要增加对netease 云硬盘的支持，包括挂载和卸载操作。同时需要修改调用docker接口进行创建和恢复容器的参数和流程。

### 快照

后续补充

# 通用关键技术

无

# 系统api接口变更

## 变更api描述

|  |  |  |
| --- | --- | --- |
| 操作 | 接口 | 描述 |
| 新增 |  |  |
| 修改 |  |  |
| 删除 |  |  |

## Api变更详情

对用户没有接口变更

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| http url | | | | |
|  | | | | |
| parameters | | | | |
| prameter | value | description | Parameter type | Data type |
| pretty |  | True:output is pretty printed | query | string |
| labelSelector |  |  | query | string |
| fieldSelector |  |  | query | string |
| watch | true/false |  | query | bool |
| resourceVersion |  |  | query | sting |
| namespace |  |  | path | string |
| response | | | | |
|  | | | | |
| others | | | | |
|  | | | | |

# 模块设计

## controller模块

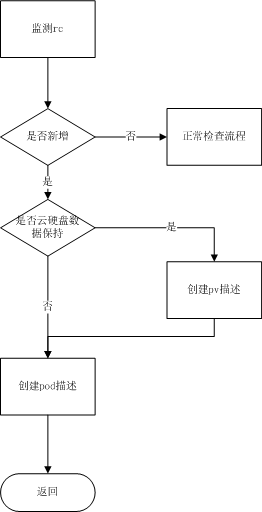
涉及到rc congroller、pv congroller和node controller。

### 关键数据设计

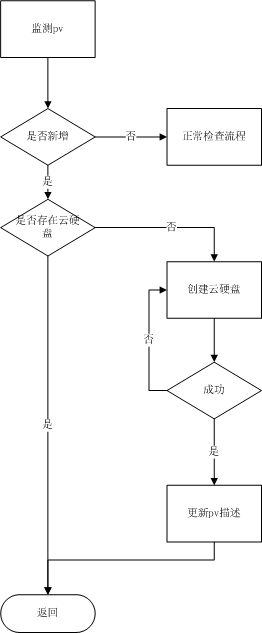
### 模块接口（不强求）

### 模块处理流程图

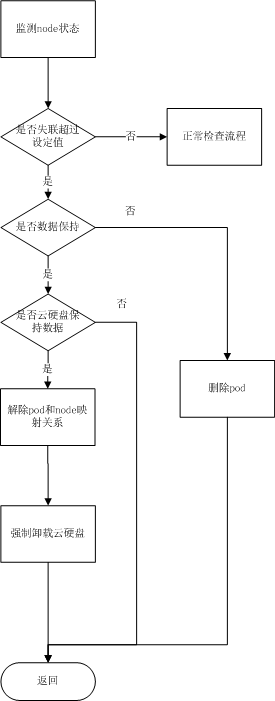
1. rc controller



1. pv controller



1. node controller



### 映射需求说明

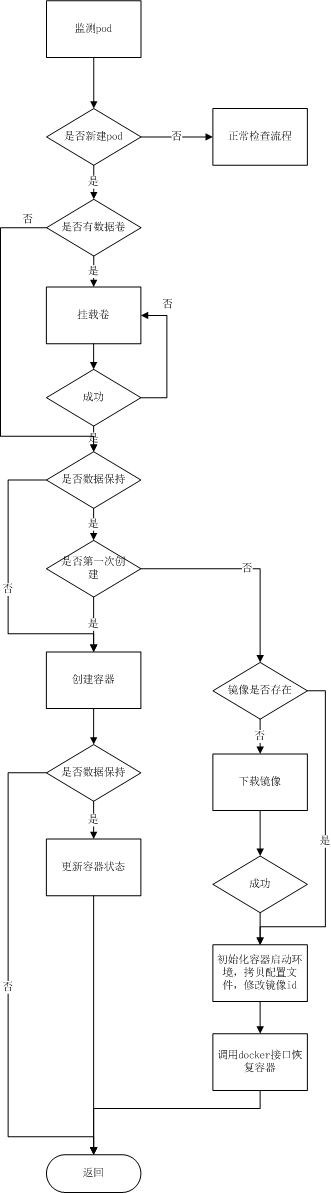
|  |  |
| --- | --- |
| 编号 | 描述 |
| K8s-sjfl-fun-1 | 支持将容器数据和volume数据保持（默认云硬盘） |
| K8s-sjfl-fun-2 | 支持将数据保持到本地盘和云硬盘两种模式 |
| K8s-sjfl-fun-4 | 支持容器重启后自动从老数据恢复 |
| K8s-sjfl-fun-5 | 数据保持盘支持垂直扩容 |

## Scheduler

因为依赖的处理流程在早期的数据保持到本地盘已经支持，无需额外增加

## kubelet

### 模块处理流程图



### 映射需求说明

|  |  |
| --- | --- |
| 编号 | 描述 |
| K8s-sjfl-fun-1 | 支持将容器数据和volume数据保持（默认云硬盘） |
| K8s-sjfl-fun-2 | 支持将数据保持到本地盘和云硬盘两种模式 |
| K8s-sjfl-fun-3 | 支持将用户-v指定的数据盘按照默认盘处理 |
| K8s-sjfl-fun-4 | 支持容器重启后自动从老数据恢复 |

# 业务场景描述

Hos

## 第一次创建rc

场景：用户创建容器

1. 创建用rc，选择schedulePolicy为monopoly，startPolicy为scheduleOld，选择磁盘大小为30G
2. Apiserver创建rc，并持久化到etcd
3. Rc congroller通过list watch得知新建rc，根据其描述startPolicy为scheduleOld判断需要创建云硬盘，因此创建了一个pv资源，大小为30G，然后根据rc创建了pod资源
4. Pv controller通过list watch得知新建了pv资源，然后调用管理平台接口在该租户下创建了一个大小30G的云硬盘，并将云硬盘uuid更新到pv资源描述中，如果创建云硬盘失败，pv controller会周期性重试，直到成功或者rc被删除
5. 与4同步，scheduler通过list watch得知有新的pod创建，scheduler将其调度到一个可用的node上，根据其schedulerPolicy判断是一对一独占方式，然后给node打上被调度pod的标签
6. 被调度的node之上kubelet通过list watch检测到新的pod被调度到该节点上，检测到pod关联了一个pv资源，这时会获取pv资源的详细情况，如果成功获得pv关联的云硬盘uuid则进行第7步，如果没有则等待下一次重试。
7. 调用volume接口将uuid对应的云硬盘挂载到node节点上，如果失败则等待下一次尝试。如果挂载成功，进入第8步。
8. 检测到需要数据保持，此时会检测是否第一次创建容器，如果是则新建容器。然后将新创建的容器信息更新到pod资源描述中。创建流程完毕。

## 容器挂掉

场景：容器被杀死，可以是人为、docker damon重启等

1. 老的rc，选择schedulePolicy为monopoly，startPolicy为scheduleOld
2. Kubelet检测到容器挂掉，判断出容器采用了数据保持，此时不会新创建容器，而是会根据老的容器id重新恢复容器

## node挂掉

场景：容器被杀死，可以是人为、docker damon重启等

1. 老的rc，选择schedulePolicy为monopoly，startPolicy为scheduleOld
2. Node controller检测到node失联超过设定值，此时获知调度到该node上容器采用云硬盘进行数据保持，node controller解除掉pod和node之间的关联
3. 为了防止原有node仍然存活，只是中间网络出现异常，此时会将原有node节点强制重启
4. 调度器检测到解绑后的pod，重新调度到一个新的可用的node节点上
5. node上的kubelet检测到新的pod，接着检测到pod关联了pv数据，会调用volume接口挂载云硬盘，如果挂载失败，会等待下一次重试
6. 挂载成功后，检测到pod采用了数据保持，进一步检测到容器已经创建过，此时会提取容器id、容器镜像等信息
7. 容器被调度到新的node节点，镜像不一定存在，此时检测镜像是否存在，如果不存在则会下载，下载失败会周期尝试。下载成功或者已经存在镜像，进入第8步
8. 此时会将容器的config配置文件拷贝到docker目录下，并更改配置文件中镜像的id
9. 调用docker接口，传入老的容器id，恢复容器

# 运维接口

### 接口说明

采用现有kubernetes命令行工具和云硬盘接口

### 运维场景及流程

1. 如果某种情况下云硬盘不能卸载成功，人为介入成功卸载云硬盘后，容器流程会自动恢复
2. 如果某种异常导致程序创建云硬盘失败，可认为介入创建云硬盘，然后手动通过kubectl更新pv描述，也能使流程成功

# 非功能性设计

## 性能

1. 根据云硬盘的测试结果，创建云硬盘很快，能在2s左右完成，controller设计创建云硬盘和创建pod并行进行，对性能影响会极小甚至忽略不计
2. 挂载云硬盘在kubelet启动容器过程中为串行，会增加启动时间，后面会将挂载和拉取镜像进行并行化优化处理，挂载云硬盘的时间根据测试结果反馈在3s左右，和拉取镜像并行化，挂载的时间影响将会忽略不计。优化将在下个迭代中进行。

## 可用性/可靠性

创建、挂载云硬盘采用重试直至成功或者删除为止，至于重试成功率基于底层服务保证。

## 可扩展性

挂载云硬盘的操作基于集群扩展带来的性能消耗并没有设计到congroller控制端，以期在集群规模扩展时，将性能消耗均衡到node节点。

## 可维护性